بسم الله الرحمن الرحيم

Islamic University – Gaza
Faculty of Information Technology
Department of Software Development

الجامعة الإسلامية – غزة
كلية تكنولوجيا المعلومات
قسم تطوير البرمجيات

Web Application for Super GYM Club
تطبيق ويب لنادي سوبر جيم

A Graduation Project Presented to the
Faculty of Information Technology, the Islamic University of Gaza
In Partial Fulfillment for the Degree of Bachelor in Information Technology

By:

Muhammad Riyad Al-Haddad - 120190922
Belal Yasin Abu Hatab - 120190733
Fadi Mohammad Abu Hasira - 120190580
Amjad Adham Abu Qaren - 120190762

Dr. Motaz Saad

Aug 2023

Abstract

A super gym owner faced a number of challenges, such as limited accessibility to gym services outside of physical premises, difficulties in managing member registrations and payments, and a lack of personalized training programs. In agreement with the gym owner, we decided to address these problems by designing a system with a unique and customized feature.

The system was implemented using the waterfall methodology. The methodology begins with the requirements phase, in which the team gathers as much information as possible to ensure a successful project. The next phase is the system design phase, in which the team designs the system architecture and functionality. The implementation phase follows, in which the team builds the system according to the design. The testing phase then ensures that the system meets the requirements. Finally, the system is deployed and maintained.

The system was completed within a six-month timeframe and was implemented in two phases. The first phase focused on the development of the core features of the system, such as online class bookings and personalized training programs. The second phase focused on the development of additional features, such as progress tracking and interactive communication channels.

The system had a significant impact on the traditional gym industry. It made it easier for people to access gym services, regardless of their location. It also helped gym owners to streamline their operations and improve the overall customer experience.

The system was a valuable initiative that had the potential to improve the gym experience for both gym owners and members. The system was well-scoped and the methodology used was appropriate for the project's unique requirements. The project was successful and had a positive impact on the traditional gym industry.

الملخص

واجه مالك صالة ألعاب رياضية كبيرة عددًا من التحديات، مثل محدودية إمكانية الوصول إلى خدمات صالة الألعاب الرياضية خارج المباني المادية، وصعوبات في إدارة تسجيلات الأعضاء والمدفوعات، ونقص برامج التدريب الشخصية. وبالاتفاق مع مالك صالة الألعاب الرياضية، قررنا معالجة هذه المشكلات من خلال تصميم نظام بميزات فريدة ومخصصة.

تم تنفيذ النظام باستخدام منهجية الشلال. تبدأ المنهجية بمرحلة المتطلبات، حيث يقوم الفريق بجمع أكبر قدر ممكن من المعلومات لضمان نجاح المشروع. المرحلة التالية هي مرحلة تصميم النظام، حيث يقوم الفريق بتصميم بنية النظام ووظائفه. تتبع ذلك مرحلة التنفيذ، حيث يقوم الفريق ببناء النظام حسب التصميم. وتضمن مرحلة الاختبار بعد ذلك أن النظام يلبي المتطلبات. وأخيراً، تم نشر النظام وصيانته.

تم الانتهاء من النظام خلال فترة زمنية مدتها ستة أشهر وتم تنفيذه على مرحلتين. ركزت المرحلة الأولى على تطوير الميزات الأساسية للنظام، مثل حجز الفصول الدراسية عبر الإنترنت وبرامج التدريب الشخصية. وركزت المرحلة الثانية على تطوير ميزات إضافية، مثل تتبع التقدم وقنوات الاتصال التفاعلية.

كان للنظام تأثير كبير على صناعة الصالات الرياضية التقليدية. إنه يسهل على الأشخاص الوصول إلى خدمات الصالة الرياضية، بغض النظر عن موقعهم. كما ساعد أصحاب الصالات الرياضية على تبسيط عملياتهم وتحسين تجربة العملاء بشكل عام.

كان النظام بمثابة مبادرة قيمة لديها القدرة على تحسين تجربة الصالة الرياضية لكل من أصحاب وأعضاء الصالة الرياضية. كان النظام واسع النطاق وكانت المنهجية المستخدمة مناسبة لمتطلبات المشروع الفريدة. كان المشروع ناجحًا وكان له تأثير إيجابي على صناعة الصالات الرياضية التقليدية.

Table of Contents

Table of Figures

Table of Tables

Introduction

The fitness industry has undergone a significant digital transformation in recent years. However, many traditional gyms still face challenges in providing a seamless and efficient online experience for their members. These challenges include limited accessibility to gym services outside of physical premises, difficulties in managing member registrations and payments, and a lack of personalized training programs.

To address these challenges and enhance the overall user experience for both gym owners and members, we propose to develop a web-based gym system that seamlessly integrates various aspects of gym management and member engagement. The system will incorporate features such as online class bookings, personalized workout plans, progress tracking, secure payment processing, and interactive communication channels between trainers and members.

The web-based gym system project is well-scoped and the methodology used is appropriate for the project's unique requirements. The methodology follows a sequential approach, specifically Waterfall, which is a sequential approach to software development. The methodology begins with the requirements phase, in which the team gathers as much information as possible to ensure a successful project. The next phase is the system design phase, in which the team designs the system architecture and functionality. The implementation phase follows, in which the team builds the system according to the design. The testing phase then ensures that the system meets the requirements. Finally, the system is deployed and maintained.

The web-based gym system project is a valuable initiative that has the potential to improve the gym experience for both gym owners and members. The project will enhance the user experience, attract and retain members, and deliver tailored services, benefiting both gym owners and members alike.

Statement of the problem

The fitness industry has witnessed a significant shift toward digital solutions in recent years. However, many traditional gyms especially in Gaza Strip establishments still face challenges in providing a seamless and efficient online experience for their members. These challenges include limited accessibility to gym services outside of physical premises, difficulties in managing member registrations and payments, and a lack of personalized training programs. Consequently, there is a need for a comprehensive web-based gym system that addresses these issues and enhances the overall user experience for both gym owners and members.

In order to cater to the evolving needs of fitness enthusiasts and gym owners alike, it is imperative to develop a robust web-based gym system that seamlessly integrates various aspects of gym management and member engagement. This system should offer features such as online class bookings, personalized workout plans, progress tracking, secure payment processing, and interactive communication channels between trainers and members.

By addressing these challenges and providing a user-friendly platform, the web-based gym system will enable gym owners to expand their reach beyond physical locations, attract and retain members more effectively, streamline administrative tasks, and ultimately enhance the overall gym experience for both parties involved. Therefore, it is crucial to develop a comprehensive web-based gym system that offers a holistic solution to the existing challenges faced by traditional gym establishments in Gaza Strip.

Objectives

Main objective

The main objective of the web-based gym system project is to develop and implement a comprehensive online platform that addresses the challenges faced by traditional gym

establishments, enhances the user experience for gym owners and members, and provides seamless accessibility to gym services beyond physical premises.

Specific objectives

The specific objectives of the project are:

- To design and develop an intuitive and user-friendly web-based interface that allows members to easily navigate and access various gym services and features.
- To integrate a personalized training program functionality that offers tailored workout plans based on individual goals, preferences, and fitness levels.
- To incorporate an online class booking system that allows members to reserve and manage their attendance for group fitness sessions conveniently.
- To establish a robust progress tracking mechanism that enables members to record and monitor their fitness achievements, including workout history, measurements, and milestones.
- To provide interactive communication channels, such as messaging or chat features, between trainers and members to facilitate seamless and effective communication.
- To ensure the compatibility and responsiveness of the web-based gym system across various devices, including desktops, laptops, tablets, and mobile phones.
- To conduct thorough testing and debugging to guarantee the system's stability, reliability, and smooth performance under different usage scenarios.

Importance of the project

The web-based gym system project is important due to its potential to revolutionize the traditional gym industry and provide practical value to gym owners and members. It enhances accessibility to gym services, streamlines member management, offers personalized training programs, facilitates convenient class booking, enables progress tracking and motivation, fosters effective communication channels, and allows for scalability and future growth. Overall, the project improves the gym experience, attracts and retains members, and delivers tailored services, benefiting both gym owners and members alike.

Scope and limitations of the project

Project Scope:

The scope of the web-based gym system project encompasses the development and implementation of a comprehensive online platform that addresses the challenges faced by traditional gym establishments. The project aims to enhance the user experience for both gym owners and members by providing seamless accessibility to gym services beyond physical premises. The system will incorporate features such as online class bookings, personalized training programs, progress tracking, and interactive communication channels between trainers and members.

The project will involve the design and development of a user-friendly web-based interface, the integration of member management functionalities, the implementation of personalized training programs, the establishment of progress tracking mechanisms.

Project Limitations:

1. The project aimed for web-based compatibility across various devices, prioritizing desktops, laptops, tablets, and mobile phones. The mobile version

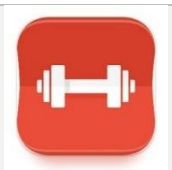
was a potential future consideration, with the primary focus on delivering a robust web-based gym system accessible on a wide range of devices. While there were potential compatibility issues with older or less common devices, the system emphasized compatibility with widely used platforms, and the mobile release was a vision for the future.

2. In its initial phase, the system lacked online payment methods, requiring users to pay for gym memberships and services in cash. However, plans were in place to introduce electronic payment options, such as credit card processing and online payment gateways, in future system updates. During this phase, cash transactions were the sole payment method, with electronic options slated for integration in subsequent releases.
3. Scalability beyond Initial Deployment: The project was designed to accommodate a small number of users within the initial deployment phase. Extensive scalability measures, such as load balancing or cloud infrastructure provisioning, required additional considerations and efforts beyond the scope of the project's initial phase.

State of the art and review of related works

The comparison between our system and related works was done after downloading and testing the applications, and the comparisons were determined based on extensive research on the most important common features required between sports programs and their users, which reached the following features:

Table 1: Comparison among related works & our system

Brand Name	Oxygen Gaza App	Fitness & Bodybuilding App	Home Workout - No Equipment App	Super Gym
Logo				
Type	Mobile	Mobile	Mobile	Web
User Interface	3 / 5	2 / 5	2 / 5	4 / 5
Exercises Collection	No	Yes	Yes	No
Booking Session	Yes	No	No	Yes
Blogs	No	No	No	Yes

About Each App According to Google Store:

About Oxygen Gaza App:

Oxygen Gym, a renowned fitness center in Gaza, distinguishes itself with its vast member community and personalized health and wellness programs. Their dedicated team crafts customized plans for each member, focusing on overall improvement in health and fitness. To enhance member experiences, they've introduced a cutting-edge mobile app that offers updates on gym activities and facilitates progress tracking, ultimately contributing to better health outcomes for all members. [1]

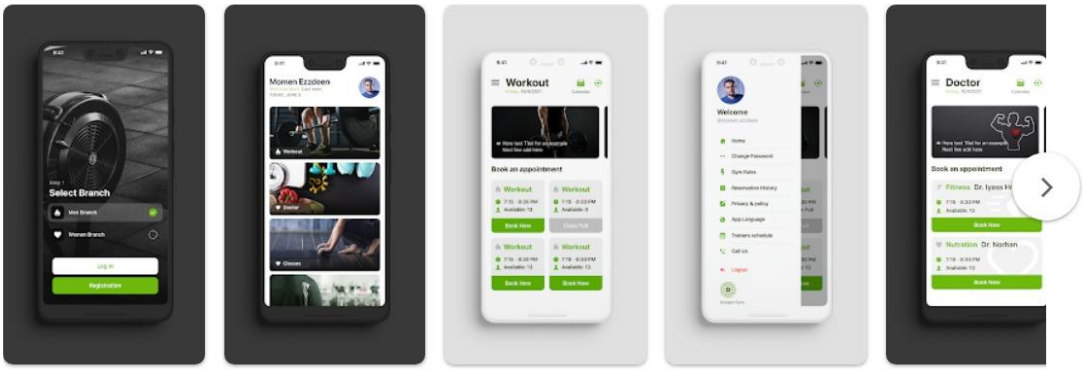


Figure 1: Oxygen App UI Samples

About Fitness & Bodybuilding App:

The Fitness & Bodybuilding app is your ultimate fitness companion, offering a wide range of workout plans tailored to your fitness goals, whether it's bodybuilding, strength-training, muscle tone, general conditioning, or powerlifting. Its intuitive interface makes it easy for users to build core strength, shed unwanted pounds, and boost overall performance. Acting as a virtual personal trainer, this app enables you to monitor your progress, analyze your workouts effectively, and receive constant guidance and support on your fitness journey. [2]

Figure 2: Fitness App UI Samples

About Home Workout - No Equipment App:

The Home Workouts app is your go-to solution for achieving your fitness goals without the need for a gym or expensive equipment. With daily routines that target major muscle groups, you can build muscle and maintain your fitness right in the comfort of your own home. Expertly designed workouts for abs, chest, legs, arms, butt, and full body are at your fingertips, and the best part is, they require no special equipment or personal coach. These short but highly effective routines make it easy to stay in shape and reach your fitness objectives. Say goodbye to gym visits and hello to convenient, effective workouts with the Home Workouts app. [3]

Figure 3: Home Workout App UI Samples

Methodology

Project Methodology

What methodology did we choose?

The Waterfall methodology is a sequential project management approach that breaks down a project into a series of phases, each of which had to be completed before the next phase could begin. The phases of the Waterfall methodology are: [4]

1. **Requirements gathering**: We gathered the requirements for the project from the stakeholders.
2. **System design**: We designed the system to meet the requirements that we gathered in the previous phase.
3. **Development**: We developed the system based on the design that we created in the previous phase.
4. **Testing**: We tested the system to ensure that it met the requirements and that it was free of defects.
5. **Deployment**: We deployed the system to the production environment.
6. **Maintenance**: We maintained the system to ensure that it continued to meet the requirements of the stakeholders.

Why We Choose Waterfall Methodology?

We chose the Waterfall methodology for the development of our project because it was a well-defined and structured methodology that could be easily followed. We believed that this methodology would help us to ensure that the project was completed on time and within budget, and that it would minimize the risks associated with the project.

The Benefits of Waterfall Methodology.

The Waterfall methodology had a number of benefits, including:

- It was a well-defined and structured methodology that could be easily followed.
- It was a good choice for projects with well-defined requirements.
- It could help to ensure that the project was completed on time and within budget.
- It could help to minimize risks by ensuring that all requirements were met before the system was deployed.

How We Adapted Waterfall Methodology.

We adapted the Waterfall methodology to our project by:

- Allowing for more flexibility in the requirements gathering phase.
- Conducting more frequent reviews and testing throughout the development process.
- Using agile development techniques, such as continuous integration and delivery, to improve the quality of the code.

We believed that these adaptations would make the Waterfall methodology a more effective choice for our project.

Tools, equipment's and methods

Here is the software we used to complete this project:

1. **Microsoft Word**

   Microsoft Word is a popular word processing software with a user-friendly interface and a range of essential features. It allows users to create, edit, and format documents, offering options for spell check, templates, and collaboration. Word enables professional document creation, customization of layouts, and integration with other Microsoft Office applications. It is widely used for various purposes, such as writing letters, reports, resumes, and essays. [5]

2. **Microsoft Project**

   Microsoft Project is project management software developed by Microsoft, aimed at helping individuals and teams effectively plan, track, and manage projects. It enables users to create schedules, allocate resources, set task dependencies, and visualize project progress through features like Gantt charts. The software facilitates resource management, budgeting, and cost tracking, allowing for efficient project execution. Collaboration tools permit teams to work together on projects and generate various reports for insights into project performance. Microsoft Project comes in different editions and offers cloud-based solutions for online collaboration and accessibility. However, alternative project management tools also exist to cater to diverse project needs. [6]

3. **Diagrams.net**

   Diagrams.net is a web-based diagramming tool used to create various types of diagrams and flowcharts. It provides an intuitive interface, a vast library of shapes, and customization options. Users can collaborate in real-time and integrate with

popular cloud storage platforms, making it an efficient tool for visualizing complex ideas and communicating them effectively. [7]

4. **Figma (Software)**

   Figma is a cloud-based design and prototyping tool that facilitates collaborative design work. Its key features include real-time editing, prototyping capabilities, and a collaborative interface. It allows designers and teams to work together efficiently, creating interactive designs and user interfaces. Figma has gained popularity for its streamlined design workflow and its ability to create visually appealing and user-friendly designs. [8]

5. **Visual Studio Code**

   Visual Studio Code is a lightweight and popular source code editor with powerful features like extensions, debugging, code completion, and version control integration. It supports multiple programming languages and frameworks, providing developers with a user-friendly interface and enhancing their productivity and collaboration. [9]

Here is the programming languages and technologies we used to complete this project:

1. **Laravel v9.x (PHP Framework):**

   Laravel is a free and open-source PHP web framework that is easy to learn and use. It provides a wide range of features that make it well-suited for building complex web applications. Laravel is also actively maintained and updated, which means that it is a reliable framework to use for long-term projects. Some of the benefits of using Laravel include its easy-to-learn codebase, large community of users and

developers, powerful features, and active maintenance. If you are looking for a powerful and easy-to-use PHP web framework, then Laravel is a great option. [10]

2. **JavaScript (Vanilla JS):**

Vanilla JavaScript is the basic JavaScript language without any additional libraries or frameworks. It is a powerful language that can be used to create a wide variety of web applications. Vanilla JavaScript is lightweight and efficient, easy to learn and use, and flexible and extensible. However, it can be difficult to write complex code and debug code without using libraries or frameworks.

Libraries and frameworks can provide additional features and functionality to Vanilla JavaScript code, but they can also make your code more complex and difficult to maintain. The decision of whether to use Vanilla JavaScript or a library or framework depends on your specific needs. [11]

3. **Bootstrap v5.3 (CSS Framework):**

Bootstrap is a free and open-source CSS framework that can be used to create responsive, mobile-first websites. It is easy to use and provides a wide range of features, including buttons, forms, navigation, and more. Bootstrap is also well-documented and has a large community of users and developers who can help if you get stuck. However, Bootstrap can be a bit heavy and limiting, and it may not always be up-to-date with the latest browsers. [12]

4. **Cascading Style Sheet v3 (CSS 3):**

CSS3 is the third major version of the CSS standard, a style sheet language used to describe the presentation of a document written in HTML or XML. CSS3 builds on the previous versions of CSS, adding new features and capabilities. These features include new selectors, properties, and modules, which allow web developers to more precisely select elements on a web page, control the appearance of elements

in more detail, and choose the modules that they need. CSS3 is not yet fully implemented in all web browsers, but support is growing rapidly. Most modern browsers support the most important features of CSS3, and support for new features is being added all the time. CSS3 can be used to create more visually appealing, accessible, and performant web pages. [13]

5. **Hyper Text Markup Language v5 (HTML 5):**

HTML 5 is the fifth and final major version of the HTML standard, a markup language used for structuring and presenting content on the World Wide Web. HTML 5 builds on the previous versions of HTML, adding new elements and capabilities, such as new elements for semantic markup, support for embedding video and audio content directly in web pages, support for web storage, and a number of new application programming interfaces (APIs). HTML 5 is not yet fully implemented in all web browsers, but support is growing rapidly. Most modern browsers support the most important features of HTML 5, and support for new features is being added all the time. The benefits of using HTML 5 include increased semantic markup, richer multimedia experiences, and improved user experience. [14]

Here is some methodology or techniques we used to complete this project:

1. **Waterfall Methodology:**

Waterfall methodology is a sequential approach to software development. It is a linear process that begins with the requirements phase, where the team collects as much information as possible to ensure the project's success. The next stage is the system design stage, where the team designs the system architecture and functionality. This is followed by the implementation phase, where the team builds

the system as designed. The testing phase then ensures that the system meets the requirements. Finally, the system is deployed and maintained. [4]

2. MVC Pattern:

The MVC pattern in software development organizes applications into Model (logic and data), View (UI presentation), and Controller (input and coordination) components. It enhances modularity, simplifies maintenance, and supports scalability. The Model handles data, the View displays it, and the Controller manages user input. This separation streamlines development, encourages collaboration, promotes code reuse, and provides a structured approach for managing application functionality. [15]

3. Database Entity Relationships Diagram (DB ERD):

A Database Entity Relationships Diagram (DB ERD) is a visual representation showing the logical connections among entities in a database. Entities are represented as rectangles and have attributes depicted as ovals, while relationships between entities are illustrated as connecting lines with labels. ERDs are vital tools in database design for understanding data relationships and aiding in the creation of effective database structures. [16]

Time Table

This time table is based on the project methodology we chose to develop and complete this project.

The following figures represent the Gantt Chart [17] and its Time Line:

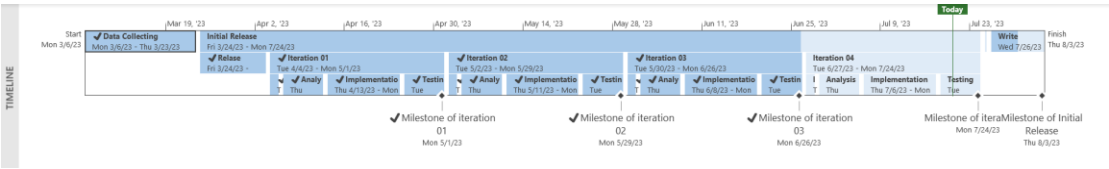
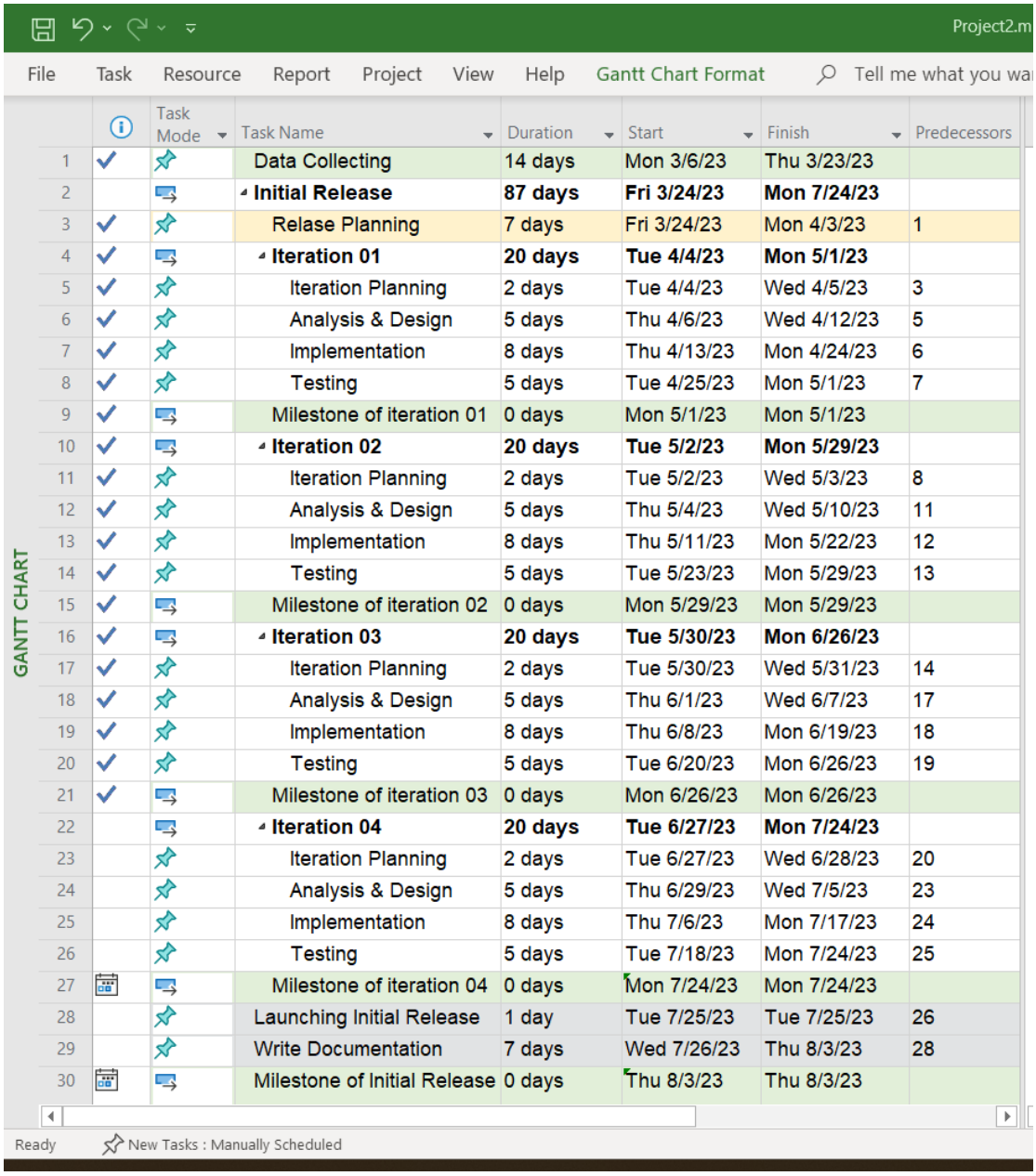


Figure 4: Gantt Chart Time Line



	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		Data Collecting	14 days	Mon 3/6/23	Thu 3/23/23	
2		**Initial Release**	**87 days**	**Fri 3/24/23**	**Mon 7/24/23**	
3		Relase Planning	7 days	Fri 3/24/23	Mon 4/3/23	1
4		**Iteration 01**	**20 days**	**Tue 4/4/23**	**Mon 5/1/23**	
5		Iteration Planning	2 days	Tue 4/4/23	Wed 4/5/23	3
6		Analysis & Design	5 days	Thu 4/6/23	Wed 4/12/23	5
7		Implementation	8 days	Thu 4/13/23	Mon 4/24/23	6
8		Testing	5 days	Tue 4/25/23	Mon 5/1/23	7
9		Milestone of iteration 01	0 days	Mon 5/1/23	Mon 5/1/23	
10		**Iteration 02**	**20 days**	**Tue 5/2/23**	**Mon 5/29/23**	
11		Iteration Planning	2 days	Tue 5/2/23	Wed 5/3/23	8
12		Analysis & Design	5 days	Thu 5/4/23	Wed 5/10/23	11
13		Implementation	8 days	Thu 5/11/23	Mon 5/22/23	12
14		Testing	5 days	Tue 5/23/23	Mon 5/29/23	13
15		Milestone of iteration 02	0 days	Mon 5/29/23	Mon 5/29/23	
16		**Iteration 03**	**20 days**	**Tue 5/30/23**	**Mon 6/26/23**	
17		Iteration Planning	2 days	Tue 5/30/23	Wed 5/31/23	14
18		Analysis & Design	5 days	Thu 6/1/23	Wed 6/7/23	17
19		Implementation	8 days	Thu 6/8/23	Mon 6/19/23	18
20		Testing	5 days	Tue 6/20/23	Mon 6/26/23	19
21		Milestone of iteration 03	0 days	Mon 6/26/23	Mon 6/26/23	
22		**Iteration 04**	**20 days**	**Tue 6/27/23**	**Mon 7/24/23**	
23		Iteration Planning	2 days	Tue 6/27/23	Wed 6/28/23	20
24		Analysis & Design	5 days	Thu 6/29/23	Wed 7/5/23	23
25		Implementation	8 days	Thu 7/6/23	Mon 7/17/23	24
26		Testing	5 days	Tue 7/18/23	Mon 7/24/23	25
27		Milestone of iteration 04	0 days	Mon 7/24/23	Mon 7/24/23	
28		Launching Initial Release	1 day	Tue 7/25/23	Tue 7/25/23	26
29		Write Documentation	7 days	Wed 7/26/23	Thu 8/3/23	28
30		Milestone of Initial Release	0 days	Thu 8/3/23	Thu 8/3/23	

Figure 5: Gantt Tasks

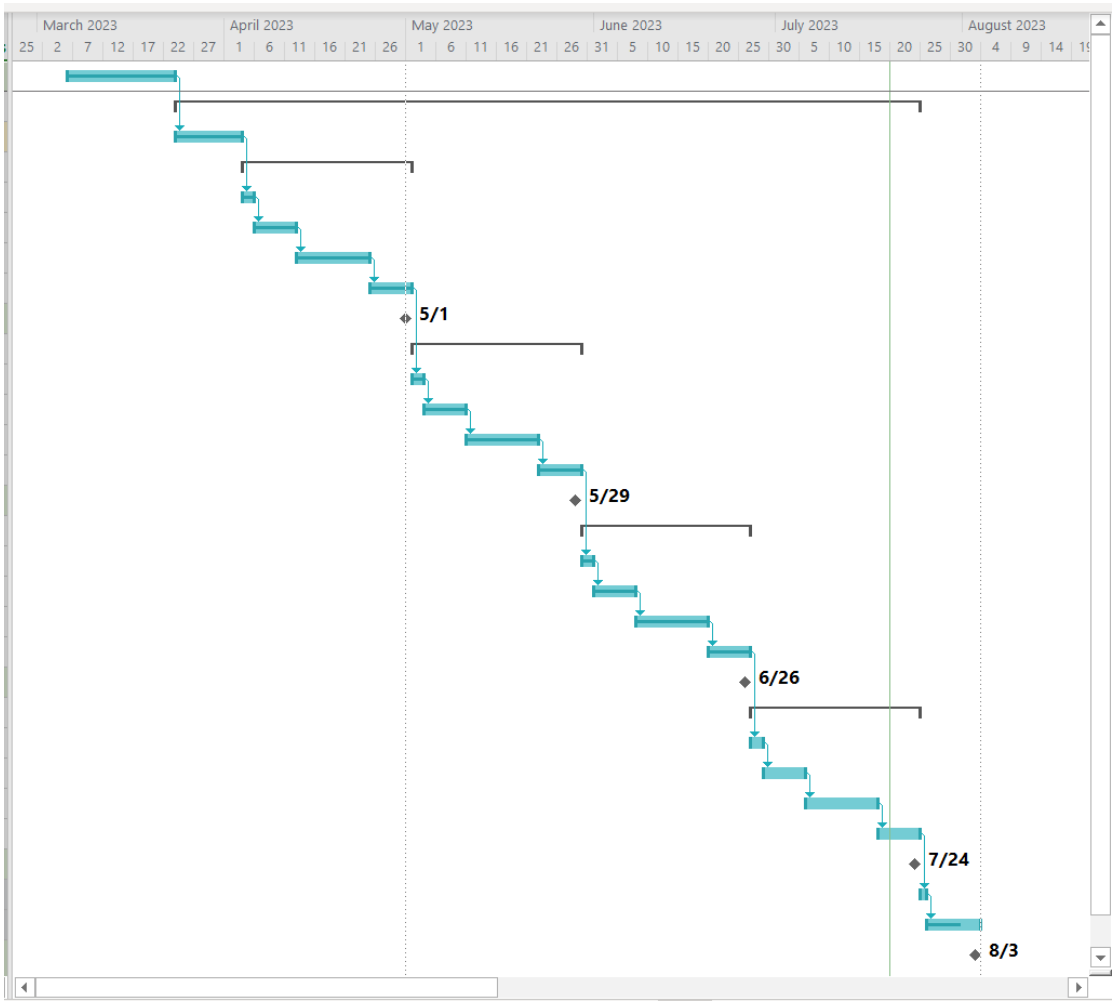
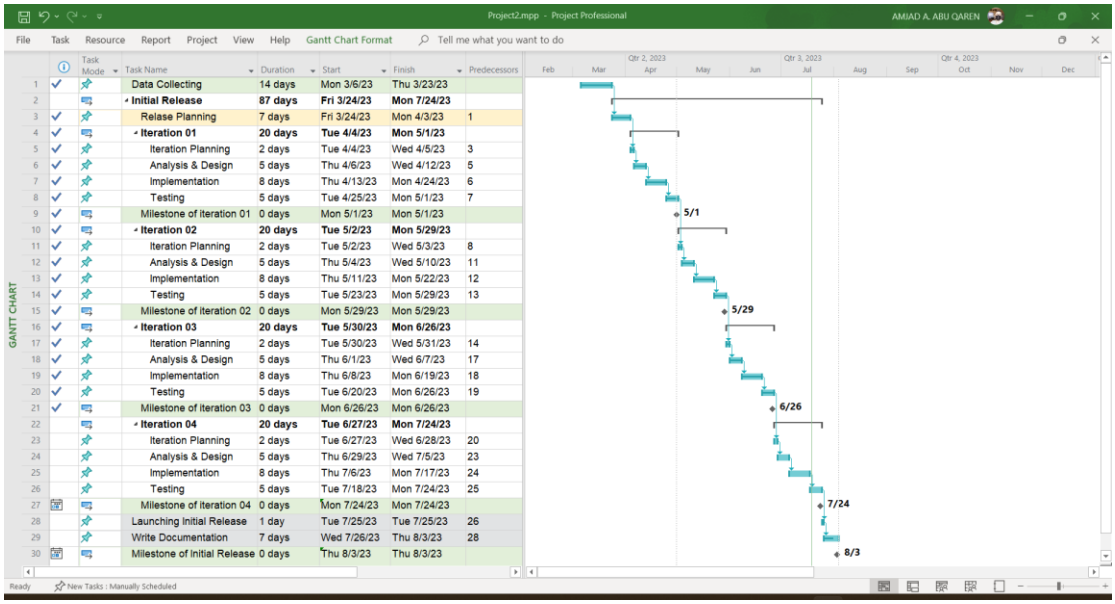


Figure 6: Gantt Chart



	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		Data Collecting	14 days	Mon 3/6/23	Thu 3/23/23	
2		**Initial Release**	**87 days**	**Fri 3/24/23**	**Mon 7/24/23**	
3		Relase Planning	7 days	Fri 3/24/23	Mon 4/3/23	1
4		**Iteration 01**	**20 days**	**Tue 4/4/23**	**Mon 5/1/23**	
5		Iteration Planning	2 days	Tue 4/4/23	Wed 4/5/23	3
6		Analysis & Design	5 days	Thu 4/6/23	Wed 4/12/23	5
7		Implementation	8 days	Thu 4/13/23	Mon 4/24/23	6
8		Testing	5 days	Tue 4/25/23	Mon 5/1/23	7
9		Milestone of iteration 01	0 days	Mon 5/1/23	Mon 5/1/23	
10		**Iteration 02**	**20 days**	**Tue 5/2/23**	**Mon 5/29/23**	
11		Iteration Planning	2 days	Tue 5/2/23	Wed 5/3/23	8
12		Analysis & Design	5 days	Thu 5/4/23	Wed 5/10/23	11
13		Implementation	8 days	Thu 5/11/23	Mon 5/22/23	12
14		Testing	5 days	Tue 5/23/23	Mon 5/29/23	13
15		Milestone of iteration 02	0 days	Mon 5/29/23	Mon 5/29/23	
16		**Iteration 03**	**20 days**	**Tue 5/30/23**	**Mon 6/26/23**	
17		Iteration Planning	2 days	Tue 5/30/23	Wed 5/31/23	14
18		Analysis & Design	5 days	Thu 6/1/23	Wed 6/7/23	17
19		Implementation	8 days	Thu 6/8/23	Mon 6/19/23	18
20		Testing	5 days	Tue 6/20/23	Mon 6/26/23	19
21		Milestone of iteration 03	0 days	Mon 6/26/23	Mon 6/26/23	
22		**Iteration 04**	**20 days**	**Tue 6/27/23**	**Mon 7/24/23**	
23		Iteration Planning	2 days	Tue 6/27/23	Wed 6/28/23	20
24		Analysis & Design	5 days	Thu 6/29/23	Wed 7/5/23	23
25		Implementation	8 days	Thu 7/6/23	Mon 7/17/23	24
26		Testing	5 days	Tue 7/18/23	Mon 7/24/23	25
27		Milestone of iteration 04	0 days	Mon 7/24/23	Mon 7/24/23	
28		Launching Initial Release	1 day	Tue 7/25/23	Tue 7/25/23	26
29		Write Documentation	7 days	Wed 7/26/23	Thu 8/3/23	28
30		Milestone of Initial Release	0 days	Thu 8/3/23	Thu 8/3/23	

Figure 7: Gantt Chart & Tasks

Analysis

System Requirements:

What are System Requirements?

System requirements, also known as software or hardware requirements, are a detailed set of specifications and criteria that outline the capabilities, constraints, and conditions necessary for a software application or system to function properly. These requirements serve as guidelines for developers, designers, and stakeholders to understand what the system needs to achieve and how it should perform. System requirements typically cover both functional and non-functional aspects of the system. Here's a breakdown of these aspects:

1. **Functional Requirements:**

   Functional requirements describe what the system should do, outlining its intended functionalities and features. These requirements specify how the system should respond to various inputs, user actions, and events. They define the interactions between users and the system and are often captured in use cases, user stories, or scenarios. Functional requirements answer questions like "What tasks should the system be able to perform?" and "What are the expected outcomes of these tasks?"

2. **Non-Functional Requirements:**

   Non-functional requirements address the qualities and characteristics that the system should possess in terms of performance, security, reliability, usability, and more. These requirements focus on how the system should perform its functions rather than just what functions it should perform. Examples of non-functional requirements include response time, scalability, security measures, accessibility, and compatibility with different devices or browsers.

Importance of System Requirements:

Defining clear and comprehensive system requirements is crucial for the success of software development projects. Properly documented requirements

help ensure that the development team and stakeholders have a shared understanding of the project's goals, functionalities, and performance expectations. Well-defined requirements also serve as a baseline for testing, validation, and quality assurance efforts. Additionally, they help manage scope by providing a reference point to determine whether a project is on track and meeting its objectives. [18]

Functional Requirements

1. User Registration and Authentication:
   - Users should be able to create accounts and log in securely.
   - User roles should include members, trainers, and administrators.
   - User authentication should support password recovery and account locking for security.

2. Membership Management:
   - The admin should be able to manage membership plans.
   - Notifications should be sent to members before their memberships expire.

3. Training Sessions:
   - Users should be able to view their training sessions.
   - The admin should be able to manage training sessions.

4. Reporting and Analytics:
   - Administrators should have access to reports on membership trends, class attendance, and financial data.
   - Trainers should be able to access reports on member progress and attendance.

Non-Functional Requirements

1. Usability and User Experience:
   - The system should have an intuitive and user-friendly interface.
   - It should be responsive and accessible from various devices and browsers.

2. Security and Privacy:
   - User data, including personal and payment information, must be securely stored and transmitted.

3. Reliability and Availability:
   - The system should have high availability to ensure users can access it at any time.
   - Regular backups and disaster recovery procedures should be in place.

4. Maintenance and Support:
   - Regular updates and maintenance should be performed to ensure the system's functionality and security.
   - Adequate user support channels should be provided for troubleshooting and inquiries.

Use Cases & Use Cases Diagram:

What is Use Case & Use Case Diagram?

Use cases and use case diagrams are concepts commonly used in software development and system analysis to define and visualize the functional requirements of a system or application. They help in understanding how users interact with the system and what actions or processes the system should support.

Use Cases:

A use case is a description of a specific interaction or scenario between a user (referred to as an actor) and a system. It outlines the steps involved in achieving a particular goal or task within the context of the system. Use cases help to capture the functional requirements and user interactions in a clear and organized manner. Each use case typically includes the following components: [19]

1. Actor: The user or external system that interacts with the system to achieve a goal.
2. Use Case Name: A descriptive name for the interaction or scenario.
3. Description: A brief overview of the purpose and goal of the interaction.
4. Preconditions: The conditions that must be met before the use case can start.
5. Steps: A sequence of actions or steps that the actor and the system take to complete the interaction.
6. Postconditions: The state of the system or any relevant outcomes after the use case is completed.

Use Case Diagrams:

A use case diagram is a graphical representation that provides an overview of the various use cases, actors, and their relationships within a system. It helps in visualizing the interactions between different actors and the system's functionalities. Use case diagrams include the following elements:

1. Actors: Represent the users, external systems, or entities interacting with the system. They are usually depicted as stick figures.
2. Use Cases: Represent the specific interactions or scenarios that users can perform within the system. These are typically depicted as ovals.
3. Relationships: Lines connecting actors and use cases to illustrate the interactions and associations. Relationships can indicate that an actor is involved in a specific use case or that an actor initiates a use case.

4. System Boundary: A rectangle that encloses all the use cases to define the scope of the system being modeled.

Use case diagrams are a valuable tool for communication between stakeholders, as they provide a high-level view of the system's functionalities and user interactions. They are used during the requirements gathering phase of software development to ensure that all necessary functionalities are considered and to facilitate a common understanding among developers, designers, and users. [20]

Use Case Diagram

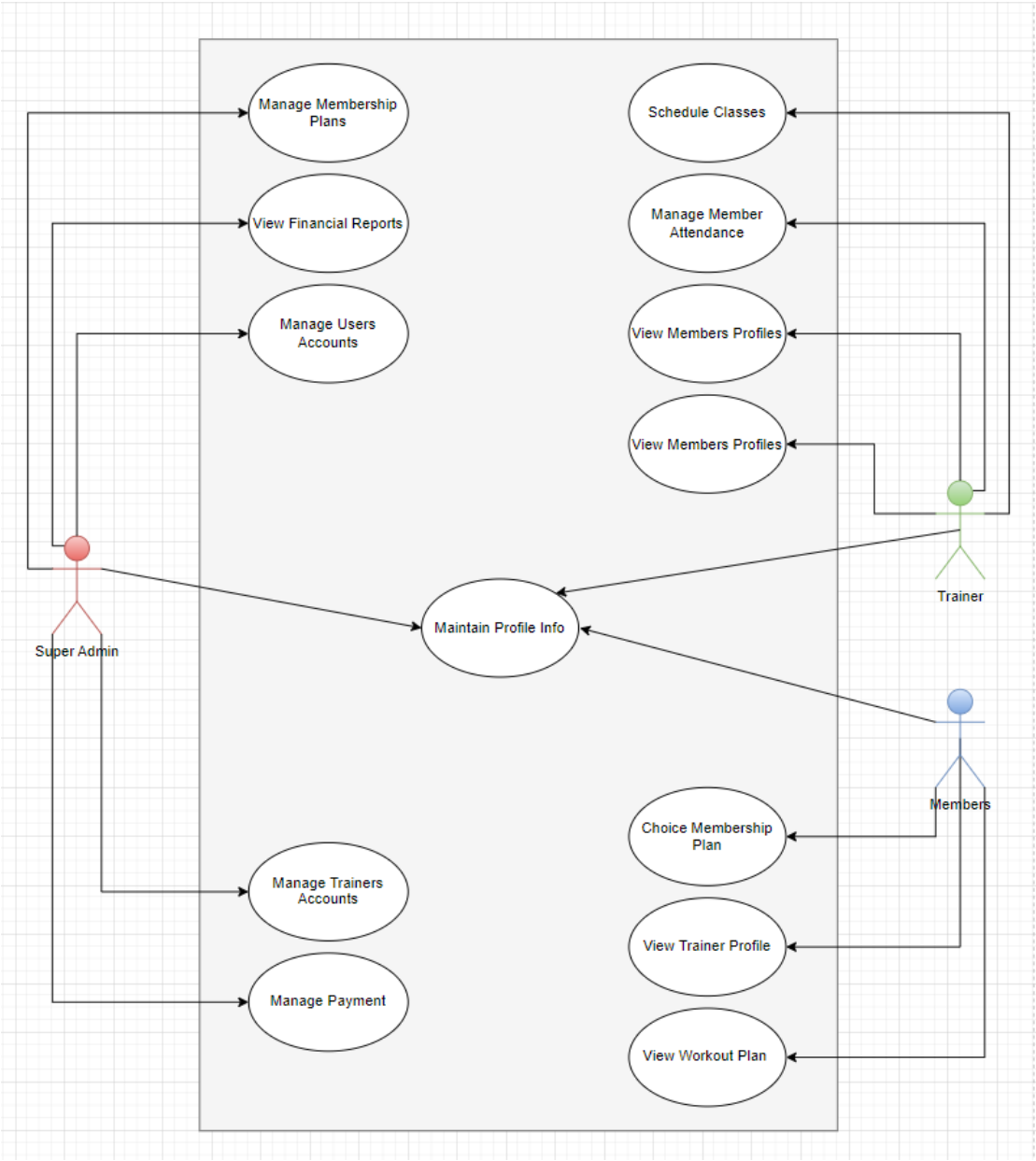


Figure 8: Use Case Diagram

Use Cases

Use Case Name:

Login

Actors:

Admin, Member, Staff

Description:

The user logs in to the gym web-based system using their username and password.

Precondition*:*

- The user has a valid username and password for the gym web-based system.

Basic Flow:

1. The user navigates to the login page of the gym web-based system.
2. The system displays the login form.
3. The user enters their username and password.
4. The user clicks the "Login" button.
5. The system verifies the user's credentials.
6. If the user is an Admin, Member, or Staff, the system redirects them to their respective dashboard.
7. If the user is a Quest User, the system redirects them to the homepage.

Alternative Flow:

1. If the user enters an incorrect username or password, the system displays an error message and prompts the user to try again.

Postcondition:

- The user is logged in to the gym web-based system and is redirected to their respective dashboard or the homepage for Quest User.

Use Case Title:

Logout

Actors:

Admin, Member, Staff, Quest User

Description:

The user logs out of the gym web-based system.

Precondition*:*

- The user is logged in to the gym web-based system.

Basic Flow:

1. The user clicks the "Logout" button.
2. The system logs the user out and redirects them to the login page.

Alternative Flow:

1. None

Postcondition:

- The user is logged out of the gym web-based system and is redirected to the login page.

Use Case Title:

Update My Profile

Actors:

Member, Staff, Admin

Description:

The user updates their profile information.

Precondition:

- The user is logged in to the gym web-based system.

Basic Flow:

1. The user navigates to their profile page.
2. The system displays the user's profile information.
3. The user clicks the "Edit" button.
4. The system displays an edit form for the user's profile information.
5. The user updates their profile information in the form.
6. The user clicks the "Save" button.
7. The system saves the updated profile information and displays a success message.

Alternative Flow:

1. If the user cancels the edit operation, the system discards any changes made to the profile information and displays the original information.

Postcondition:

- The user's profile information is updated and saved in the gym web-based system.

Use Case Title:

Change Password

Actors:

Member, Staff, Admin

Description:

The user changes their password.

Precondition:

- The user is logged in to the gym web-based system.

Basic Flow:

1. The user navigates to their account settings page.
2. The system displays the user's account information.
3. The user clicks the "Change Password" button.
4. The system displays a form for the user to enter their current password and their new password.
5. The user enters their current password and their new password.
6. The user clicks the "Save" button.
7. The system verifies that the current password is correct and saves the new password.
8. The system displays a success message.

Alternative Flow:

- If the user enters an incorrect current password, the system displays an error message and prompts the user to enter their current password again.

Postcondition:

- The user's password is changed and saved in the gym web-based system.

Use Case Title:

Choice Membership Plan

Actors:

Member

Description:

The member chooses a membership plan.

Precondition:

The member is logged in to the gym web-based system.

Basic Flow:

1. The member navigates to the "Membership Plans" page.
2. The system displays a list of available membership plans, along with their details, such as the duration, cost, and features included.
3. The member selects the desired membership plan.
4. The system prompts the member to confirm their selection.
5. The member confirms their selection.
6. The system redirects the member to the payment processing page.
7. The member enters their payment information and submits the payment.
8. The system confirms the payment and activates the chosen membership plan for the member.

Alternative Flow:

1. If the member does not wish to proceed with the payment, they can cancel the process at any time.

Postcondition:

- The member has chosen and activated a membership plan in the gym web-based system.

Use Case Title:

View Trainer Profile

Actors:

Member

Description:

The member views the profile of a trainer.

Precondition:

- The member is logged in to the gym web-based system.

Basic Flow:

1. The member navigates to the "Trainers" page.
2. The system displays a list of trainers.
3. The member selects the trainer whose profile they wish to view.
4. The system displays the selected trainer's profile, including details such as their name, photo, qualifications, and experience.
5. The member reviews the trainer's profile.

Alternative Flow:

1. If the selected trainer does not have a profile, the system displays a message indicating that no profile is available.

Postcondition:

- The member has reviewed the profile of a trainer in the gym web-based system.

Use Case Title:

View Workout Plan

Actors:

Member

Description:

The member views their assigned workout plan, which is created by the staff actor.

Precondition:

- The member is logged in to the gym web-based system and has an assigned workout plan created by the staff actor.

Basic Flow:

1. The member navigates to the "Workout Plan" page.
2. The system displays the member's assigned workout plan, which includes details such as the exercises to be performed, sets, reps, and weights.
3. The member reviews their workout plan.

Alternative Flow:

1. If the member has not yet been assigned a workout plan, the system displays a message indicating that the member needs to contact a staff member to create a plan for them.

Postcondition:

- The member has reviewed their assigned workout plan in the gym web-based system.

Use Case Title:

Manage Membership Plans

Actors:

Gym Owner

Description:

This use case describes how the gym owner can create and manage membership plans for gym members through the web-based system.

Preconditions:

- The gym owner has internet access and a valid login credential to access the gym's web-based system.
- The gym has set up their web-based system and has made it available to their members and staff.

Basic Flow:

1. The gym owner logs in to their account on the gym's web-based system.
2. The system presents the gym owner with a dashboard that displays the gym's membership plans and their current status.
3. The gym owner selects the "Manage Membership Plans" option from the dashboard.
4. The system displays a list of existing membership plans.
5. The gym owner can select an existing plan to edit or delete or create a new plan.
6. If the gym owner selects to edit a plan, the system presents a form that allows them to modify the plan details, including pricing, payment options, and benefits.
7. If the gym owner selects to delete a plan, the system presents a confirmation message to confirm the deletion of the plan.
8. If the gym owner selects to create a new plan, the system presents a form to create a new membership plan with details like the plan name, duration, pricing, payment options, and benefits.
9. The gym owner confirms the changes and saves the membership plan.

Alternate Flows:

1. If the gym owner selects to delete a membership plan that has active members, the system presents a warning message to inform the gym owner about the members affected and prompts them to confirm the deletion.
2. If the gym owner selects to create a new membership plan, the system validates the details entered, and if any information is missing, the system prompts the gym owner to fill in the required information.

Postconditions:

- The gym owner has successfully created or modified the gym's membership plans, and the changes are reflected in the system. The members will see the updated plans, and new members can join using the newly created plans.

Use Case Title:

View Statistics and Reports

Actors:

Gym Owner

Description:

This use case describes how the gym owner can view statistics and reports related to the through the web-based system.

Preconditions:

- The gym owner has internet access and a valid login credential to access the gym's web-based system.
- The gym has set up their web-based system and has made it available to their members and staff.
- Data that is already recorded in the system.

Basic Flow:

1. The gym owner logs in to their account on the gym's web-based system.
2. The system presents the gym owner with a dashboard that displays various statistics and reports related about various gym's activities.
3. The gym owner selects the "Statistics & Reports" option from the dashboard.
4. The system displays a list of statistics and reports such as financial, Status, activities report.
5. The gym owner selects the desired report to view.
6. The system presents the selected report with a date range for the report.
7. The gym owner selects the desired date range for the report.
8. The system generates the report based on the selected date range and displays it to the gym owner.
9. The gym owner can print or export the report to a file format like PDF or Excel.

Alternate Flows:

1. If the gym owner selects a report that does not have any data for the selected date range, the system displays a message informing the gym owner that there is no data available for the report.

Postconditions:

- The gym owner has successfully viewed the selected report for the gym, and they can analyze the data to make informed business decisions.

Use Case Title:

Manage Staff Accounts

Actors:

Gym Owner

Description:

This use case describes how the gym owner can create and manage staff accounts for the gym's staff members through the web-based system.

Preconditions:

- The gym owner has internet access and a valid login credential to access the gym's web-based system.
- The gym has set up their web-based system and has made it available to their members and staff.

Basic Flow:

1. The gym owner logs in to their account on the gym's web-based system.
2. The system presents the gym owner with a dashboard that displays the gym's staff and member accounts and their current status.
3. The gym owner selects the "Manage Accounts" option from the dashboard, then choice "Manage Staff Accounts" option.
4. The system displays a list of existing staff accounts.

5. The gym owner can select an existing staff account to edit or delete or create a new staff account.
6. If the gym owner selects to edit an account, the system presents a form that allows them to modify the account details, including name, contact information, role, and permissions.
7. If the gym owner selects to delete an account, the system presents a confirmation message to confirm the deletion of the staff account.
8. If the gym owner selects to create a new staff account, the system presents a form to create a new account with details like name, email, role, and permissions.
9. The gym owner confirms the changes and saves the staff account.

Alternate Flows:

1. If the gym owner selects to delete a staff account that has assigned tasks or classes, the system presents a warning message to inform the gym owner about the assigned tasks and prompts them to reassign the tasks or classes to another staff member.
2. If the gym owner selects to create a new staff account, the system validates the details entered, and if any information is missing, the system prompts the gym owner to fill in the required information.

Postconditions:

- The gym owner has successfully created or modified the gym's staff accounts, and the changes are reflected in the system. The staff members will see the updated accounts, and they can log in to the system with their assigned roles and permissions.

Use Case Title:

Manage Member Accounts

Actors:

Gym Owner

Description:

This use case describes how the gym owner can create and manage member accounts for the gym's members through the web-based system.

Preconditions:

- The gym owner has internet access and a valid login credential to access the gym's web-based system.
- The gym has set up their web-based system and has made it available to their members and staff.

Basic Flow:

1. The gym owner logs in to their account on the gym's web-based system.
2. The system presents the gym owner with a dashboard that displays the gym's staff and member accounts and their current status.
3. The gym owner selects the "Manage Accounts" option from the dashboard, then choice "Manage Member Accounts" option.
4. The system displays a list of existing member accounts.
5. The gym owner can select an existing member account to edit or delete or create a new member account.
6. If the gym owner selects to edit an account, the system presents a form that allows them to modify the account details, including name, contact information, role, and permissions.
7. If the gym owner selects to delete an account, the system presents a confirmation message to confirm the deletion of the member account.
8. If the gym owner selects to create a new member account, the system presents a form to create a new account with details like name, email, role, and permissions.
9. The gym owner confirms the changes and saves the member account.

Alternate Flows:

1. If the gym owner selects to delete a member account that currently has subscription, the system presents a warning message to inform the gym

owner about this subscription and prompts them to remind him to settle financial matters and other obligations with the member.

2. If the gym owner selects to create a new member account, the system validates the details entered, and if any information is missing, the system prompts the gym owner to fill in the required information.

Postconditions:

- The gym owner has successfully created or modified the gym's member accounts, and the changes are reflected in the system. The member will see the updated accounts, and they can log in to the system with their assigned update.

Use Case Title:

Schedule Classes

Actors:

Gym Staff (Trainer)

Description:

This use case describes how the gym staff can schedule classes for the gym members through the web-based system.

Preconditions:

- The gym staff (trainer) has internet access and a valid login credential to access the gym's web-based system.
- The gym has set up their web-based system and has made it available to their members and staff.
- The gym staff is authorized to schedule classes for the gym members.

Basic Flow:

1. The gym staff logs in to their account on the gym's web-based system.

2. The system presents the gym staff with a dashboard that displays various options related to scheduling classes.
3. The gym staff selects the "Schedule Classes" option from the dashboard.
4. The system displays a calendar with available time slots for scheduling classes.
5. The gym staff selects the desired date and time slot for the class.
6. The system prompts the gym staff to select the type of class to schedule.
7. The gym staff selects the desired class type (e.g., Yoga, Zumba, Pilates, etc.).
8. The system prompts the gym staff to enter the class details, such as class name, description, duration, and equipment needed.
9. The gym staff enters the required details and saves the class.
10. The system displays the scheduled class on the calendar.

Alternate Flows:

1. If the selected time slot is not available, the system prompts the gym staff to select a different time slot or cancel the scheduling.
2. If the gym staff enters invalid or incomplete information, the system displays an error message and prompts them to enter valid information.

Postconditions:

- The gym staff has successfully scheduled a class for the gym members, and the class is displayed on the gym's calendar.

Use Case Title:

Manage Member Attendance

Actors:

Gym Staff (Trainer)

Description:

This use case describes how the gym staff can manage member attendance for the gym classes through the web-based system.

Preconditions:

- The gym staff (trainer) has internet access and a valid login credential to access the gym's web-based system.
- The gym has set up their web-based system and has made it available to their members and staff.
- The gym staff is authorized to manage member attendance for gym classes.

Basic Flow:

1. The gym staff logs in to their account on the gym's web-based system.
2. The system presents the gym staff with a dashboard that displays various options related to managing member attendance.
3. The gym staff selects the "Manage Member Attendance" option from the dashboard.
4. The system displays a list of scheduled classes for the gym staff to select.
5. The gym staff selects the desired class for which they want to manage the attendance.
6. The system displays a list of members who have registered for the selected class.
7. The gym staff selects the member(s) who are present in the class.
8. The system updates the attendance status of the selected member(s) as "Present."
9. The gym staff can also mark absent members as "Absent."
10. The system updates the attendance status of the absent member(s) as "Absent."

Alternate Flows:

1. If the gym staff selects a class that has no registered members, the system displays a message informing them that there are no members registered for the selected class.

2. If the gym staff selects a member who is not registered for the selected class, the system displays an error message and prompts them to select a registered member.

Postconditions:

- The gym staff has successfully managed member attendance for the selected class, and the attendance records are updated in the gym's system.

Use Case Title:

View Member Profiles

Actors:

Gym Staff (Trainer)

Description:

This use case describes how the gym staff can view member profiles through the web-based system.

Preconditions:

- The gym staff (trainer) has internet access and a valid login credential to access the gym's web-based system.
- The gym has set up their web-based system and has made it available to their members and staff.
- The gym staff is authorized to view member profiles.

Basic Flow:

1. The gym staff logs in to their account on the gym's web-based system.
2. The system presents the gym staff with a dashboard that displays various options related to managing gym activities.
3. The gym staff selects the "View Member Profiles" option from the dashboard.
4. The system displays a search bar and a list of all registered gym members.

5. The gym staff can search for a specific member profile using the search bar or select a member from the list.
6. The system displays the selected member's profile page, including their personal information, fitness goals, and workout history.
7. The gym staff can view the member's attendance records, including classes they have attended and missed.
8. The gym staff can add or modify notes about the member's progress, injuries, or other relevant information.

Alternate Flows:

1. If the gym staff enters an invalid member name or ID, the system displays an error message and prompts them to enter a valid member name or ID.

Postconditions:

- The gym staff has successfully viewed the member profile, and they can use the information to tailor workout plans and provide personalized guidance to the member.

Design

System Architecture

In our project we use the Laravel MVC pattern which will impact the system structure by modularizing the application into three layers: the model, the view, and the controller. This separation of concerns will make the application easier to develop, test, and maintain.

The model layer will be responsible for encapsulating the data access logic and business rules. The view layer will be responsible for rendering the user interface. The controller layer will be responsible for routing user requests to the appropriate model and view.

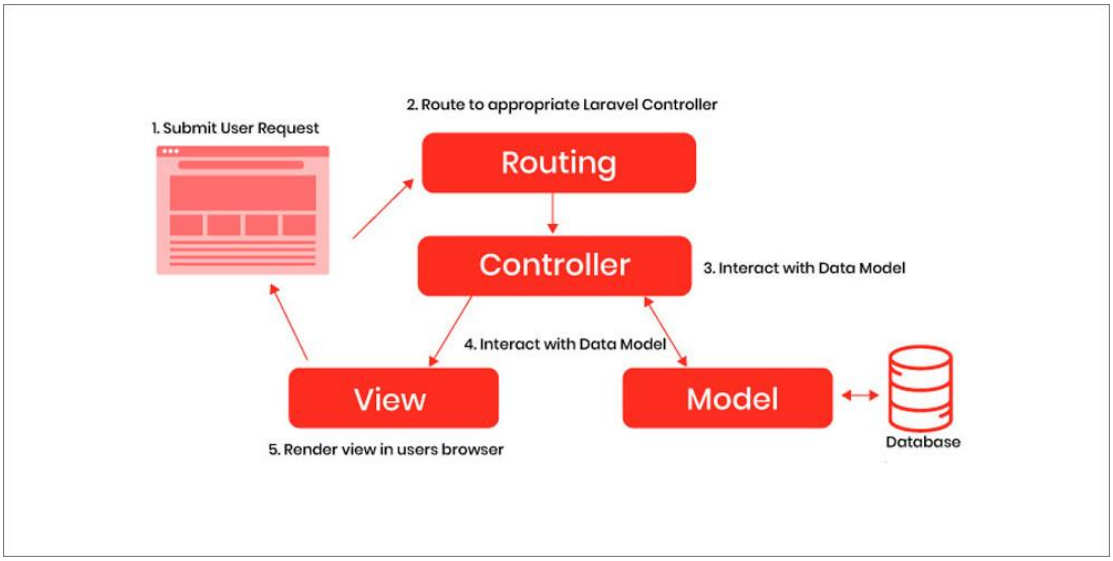


Figure 9: Laravel MVC Pattern

This separation of concerns will make the application easier to understand and maintain. It will also make it easier to test the application, as each layer can be tested independently.

In addition, the MVC pattern can help to improve the performance of the application. By isolating the database access and the user interface, the MVC pattern can help to prevent bottlenecks in the application.

Here are some additional benefits of using the Laravel MVC pattern:

1. Scalability: The application will be easier to scale as the number of users and requests increases.
2. Security: The application will be more secure by isolating the database access and the user interface.
3. Reusability: The application code will be more reusable, as the model and view layers can be reused in other applications.
4. Testability: The application will be easier to test, as each layer can be tested independently.

Incorporating the Laravel MVC pattern into the system structure will make the application more robust and reliable. It will also make the application easier to develop, test, and maintain.

Database Structure

This section shows the structure of the system database, which consists following major entities:

1. Member: This entity represents a gym member. It has the following attributes:
    - id (primary key)
    - name
    - email
    - phone number
    - address
    - date of birth
    - membership plan (foreign key)
    - role (member, trainer, administrator)
2. Membership Plan: This entity represents a gym membership plan. It has the following attributes:

- id (primary key)
- name
- price
- duration (in months)

3. Training Session: This entity represents a training session. It has the following attributes:

    - id (primary key)
    - name
    - date
    - time
    - trainer (foreign key)
    - members (many-to-many relationship)

4. Trainer: This entity represents a gym trainer. It has the following attributes:

    - id (primary key)
    - name
    - email
    - phone number
    - address

5. Admin: This entity represents a gym administrator. It has the same attributes as the Trainer entity.

The relationships between the entities are as follows:

- Member: A member can have one membership plan.
- Membership Plan: A membership plan can have many members.
- Training Session: A training session can have many members.
- Member: A member can attend many training sessions.
- Trainer: A trainer can conduct many training sessions.
- Admin: An admin can manage all the entities in the database.

The following figure show the database entity relationship diagrams (ERD):

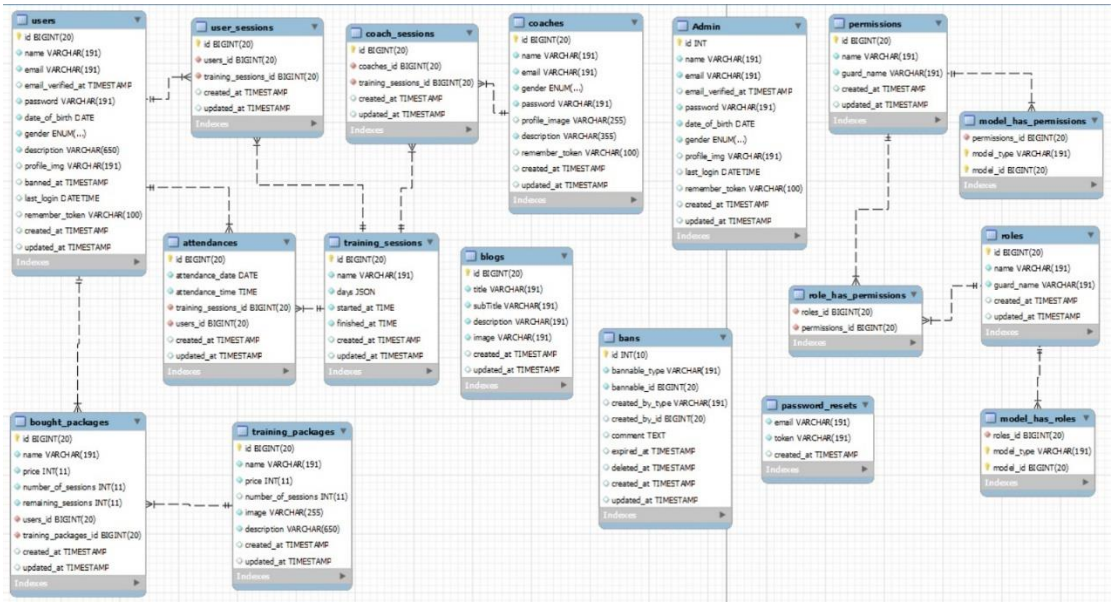


Figure 10: Database Schema

preliminary design samples

This section shows the preliminary design samples of this project.

Splash Screens:

Figure 11: Splash Screen

Figure 12: Dashboard - Splash

Sign up & Login:

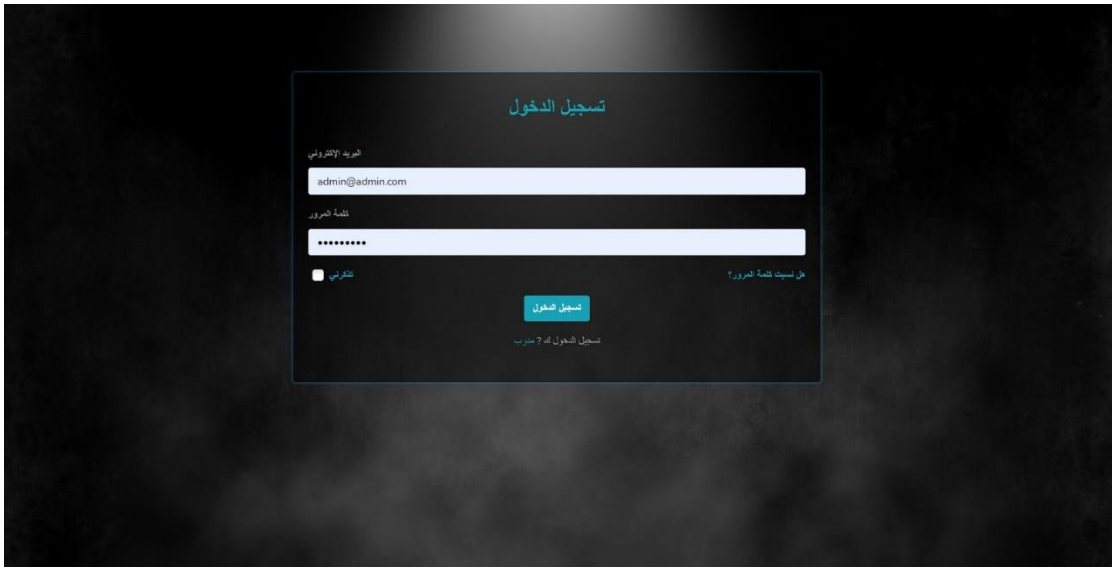


Figure 13: Dashboard - Login

Dashboard Screens:

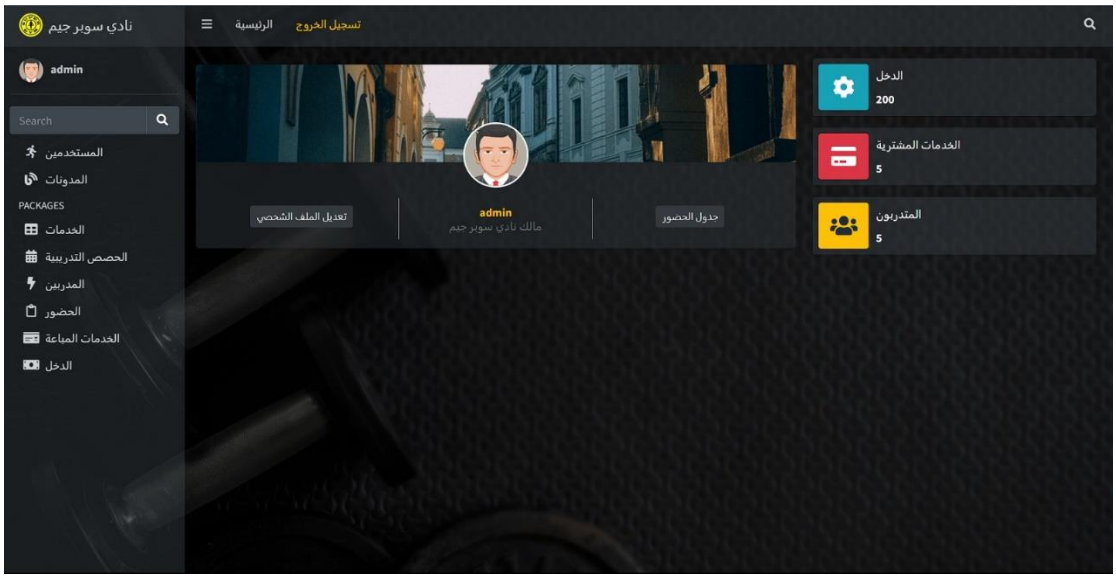


Figure 14: Dashboard - Admin - Main Layout

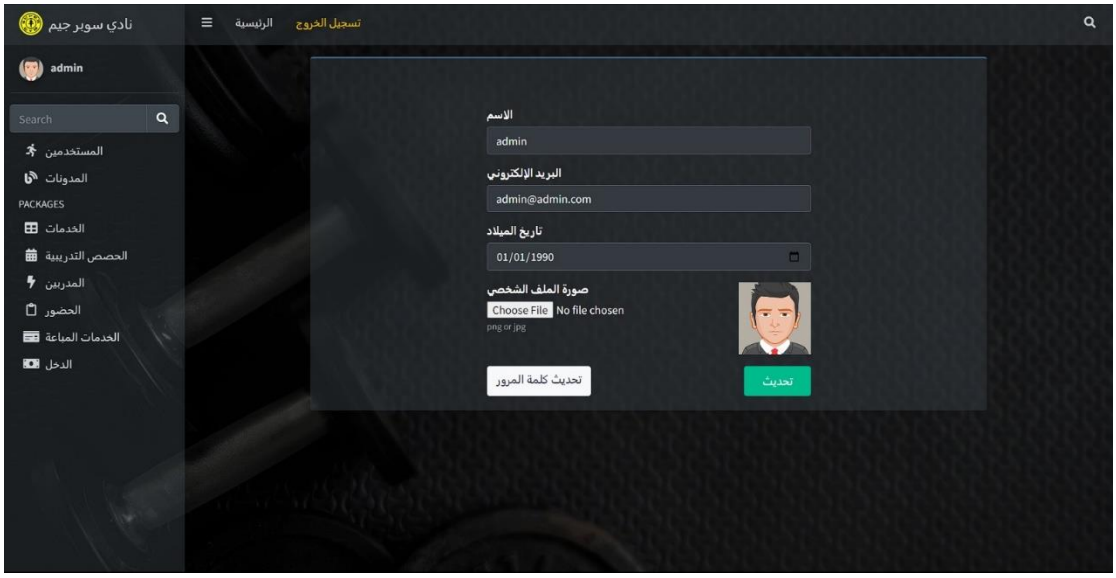


Figure 15: Dashboard - Admin - Update Profile

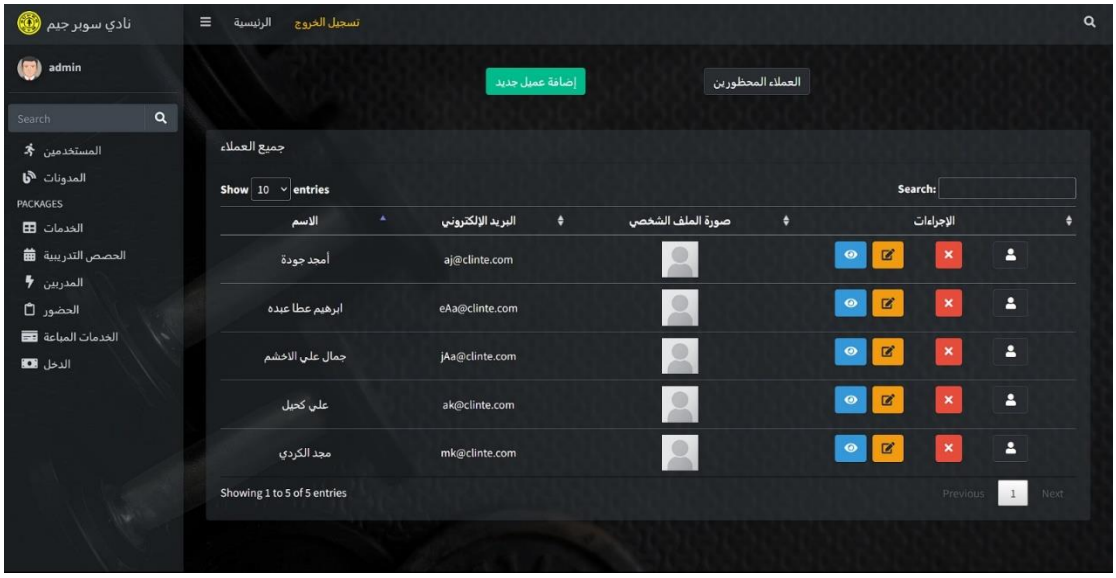


Figure 16: Dashboard - Admin - Members Table

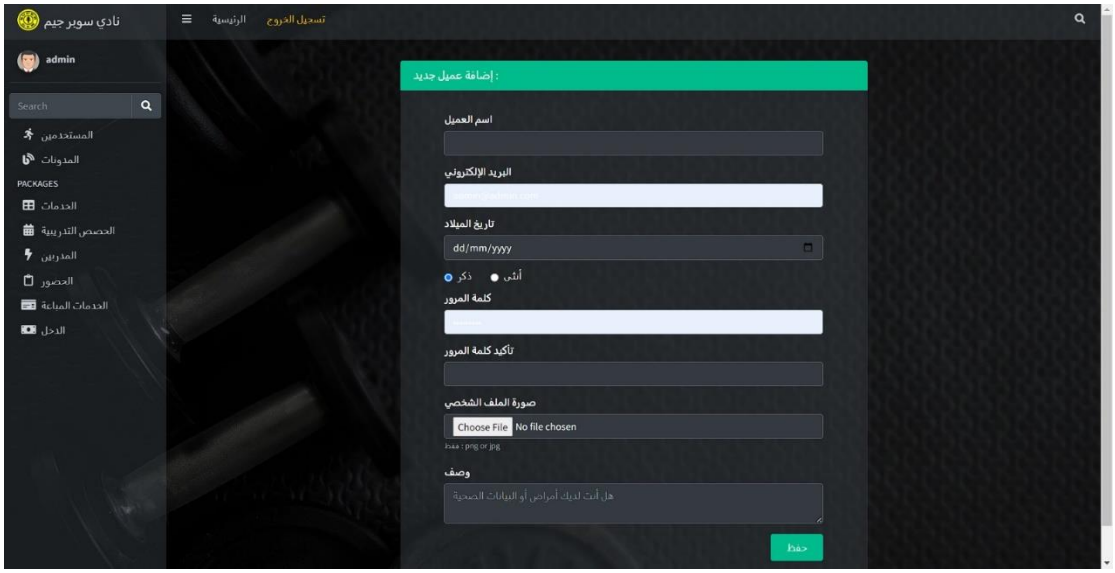


Figure 17: Dashboard - Admin - Insert a new member

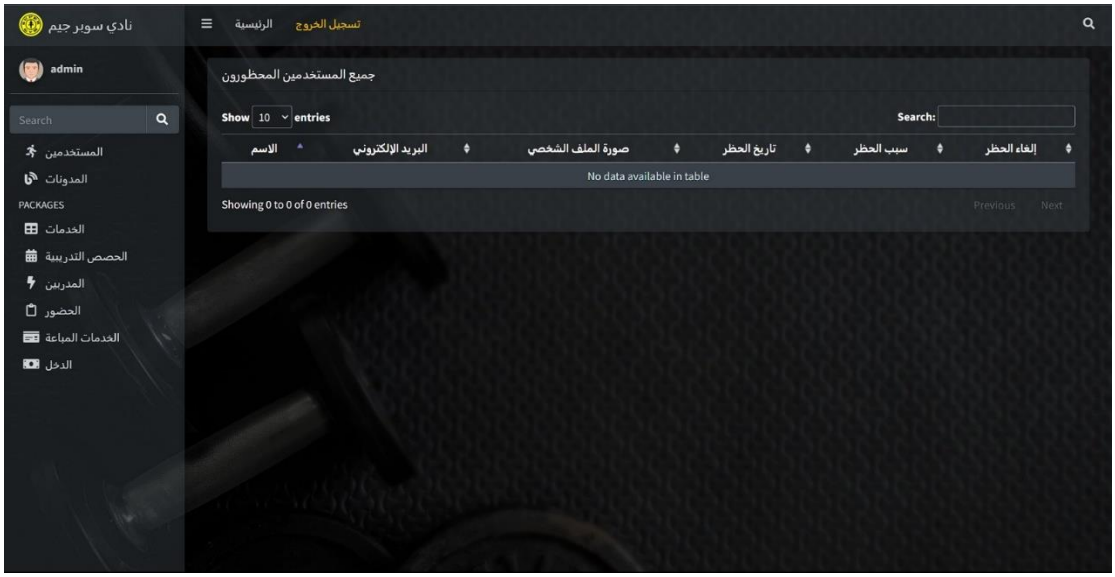


Figure 18: Dashboard - Admin - Blocked Members Table

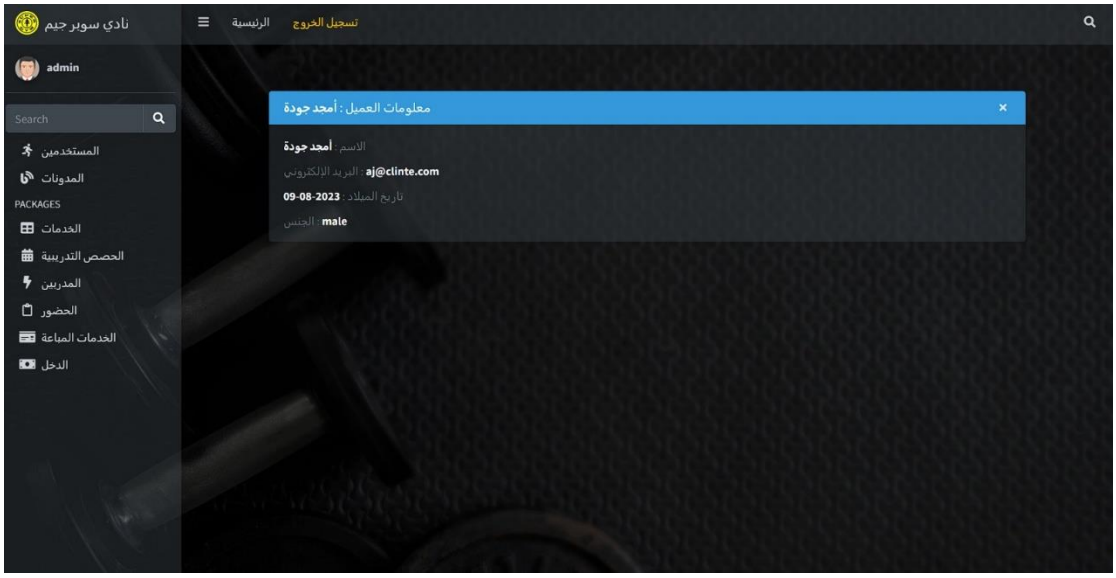


Figure 19: Dashboard - Admin - View in detailed member data

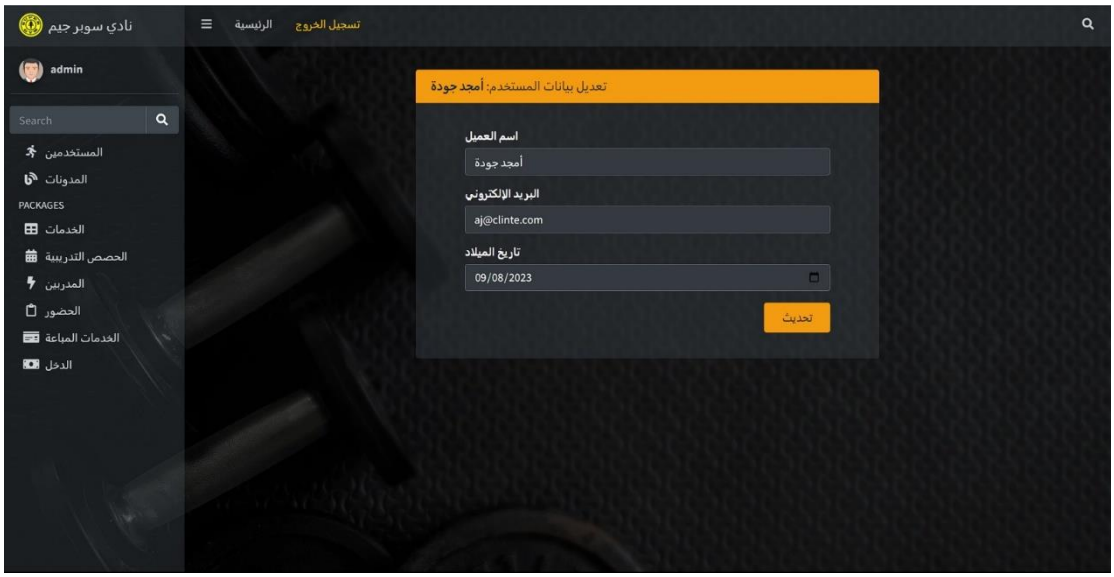


Figure 20: Dashboard - Admin - Update Member Data

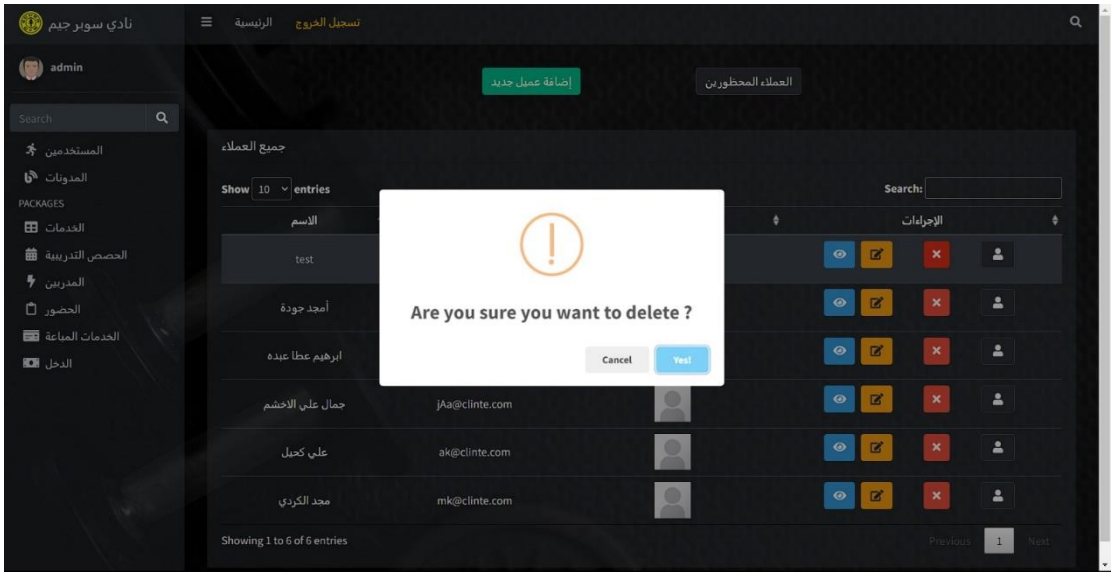


Figure 21: Dashboard - Admin - Delete Confirmation

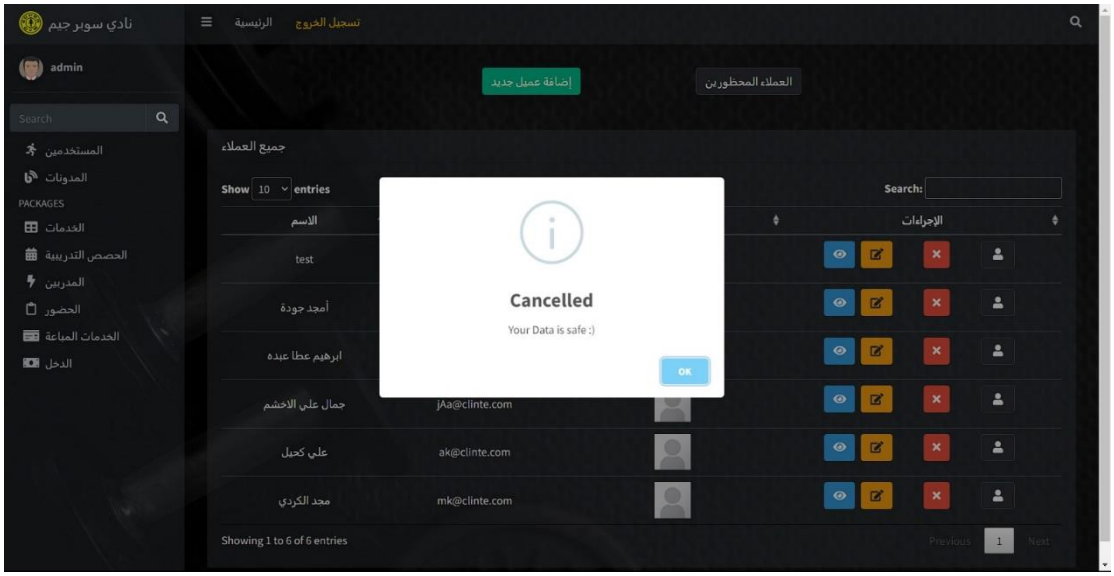


Figure 22: Dashboard - Admin - Cancel Delete Operation

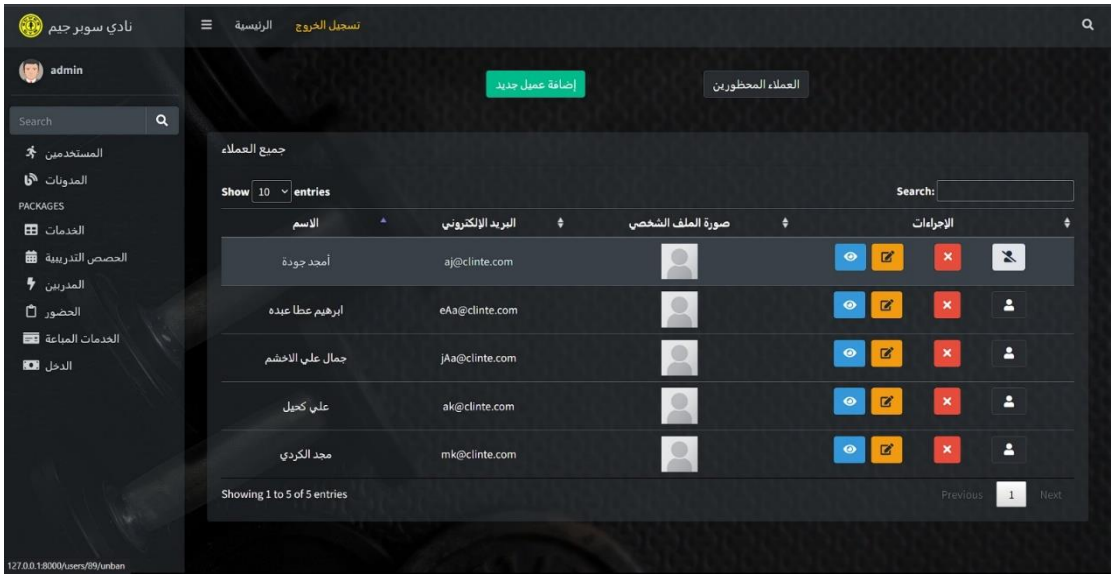


Figure 23: Dashboard - Admin - Blocking a member

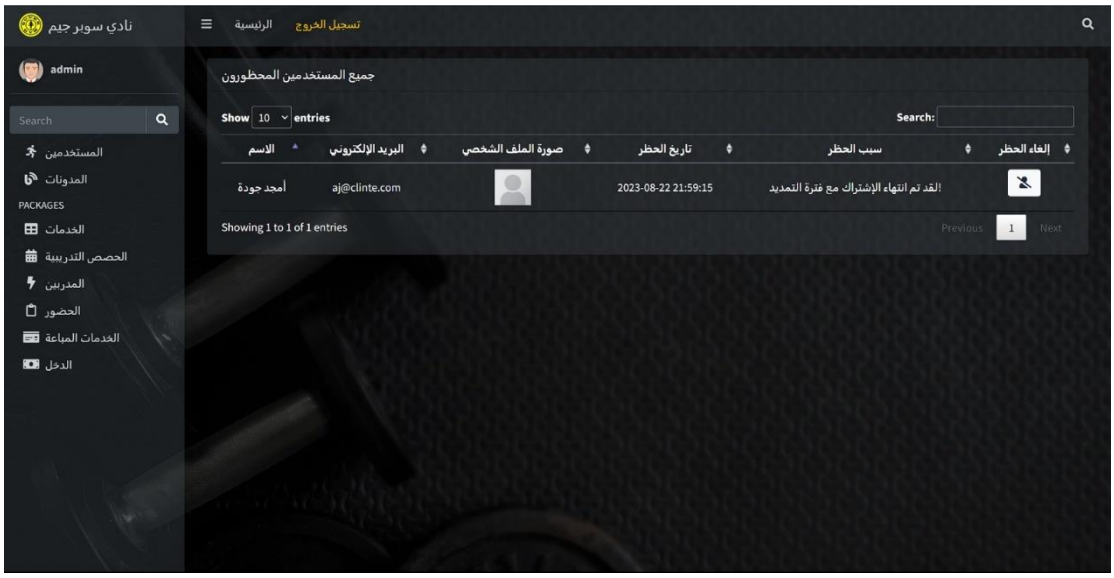


Figure 24: Dashboard - Admin - View the blocked member

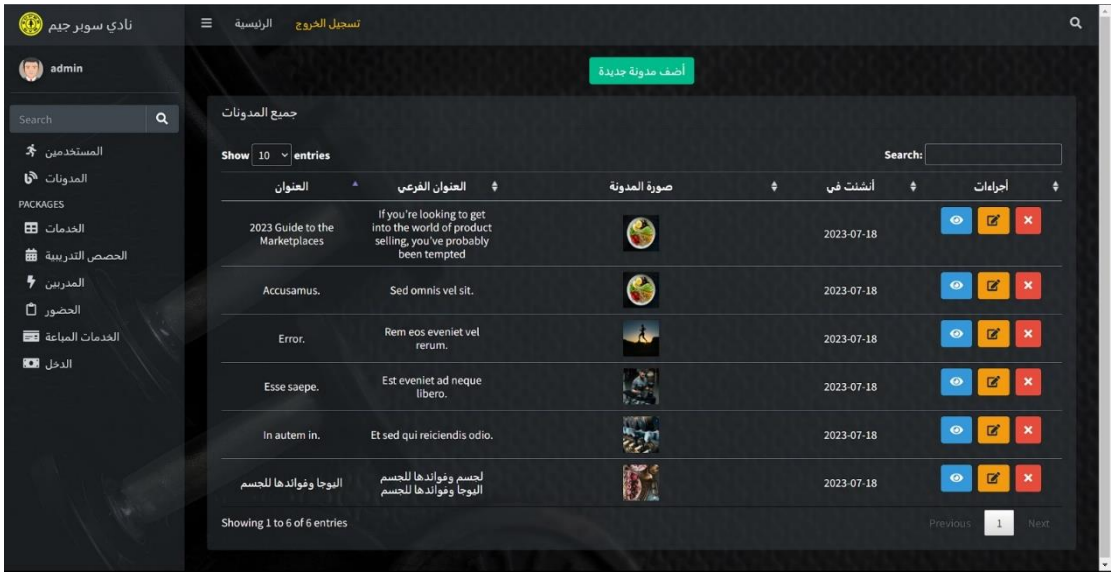


Figure 25: Dashboard - Admin - Blogs Table

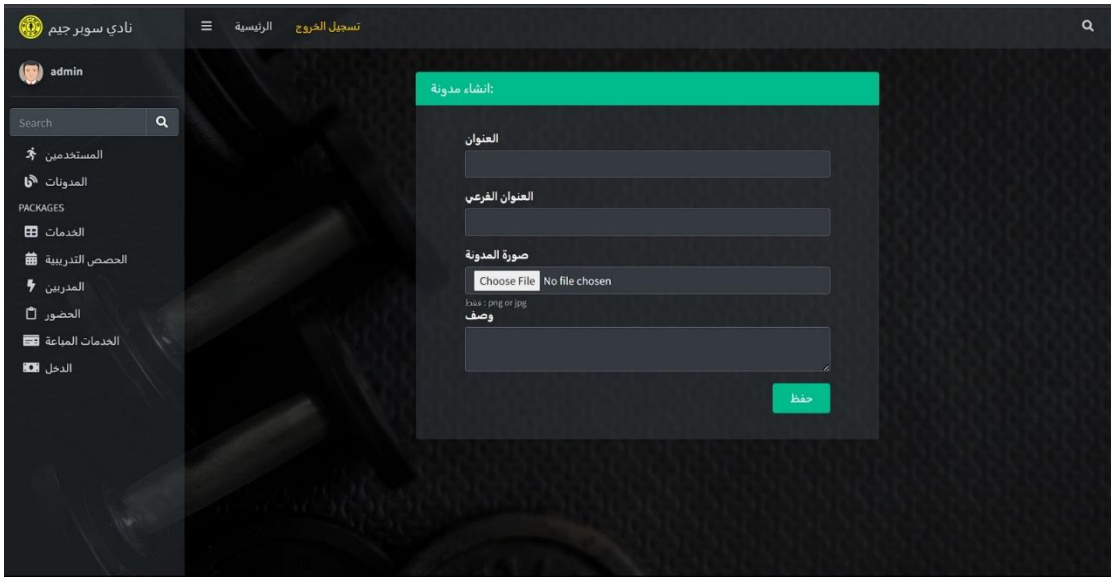


Figure 26: Dashboard - Admin - Insert a new blog

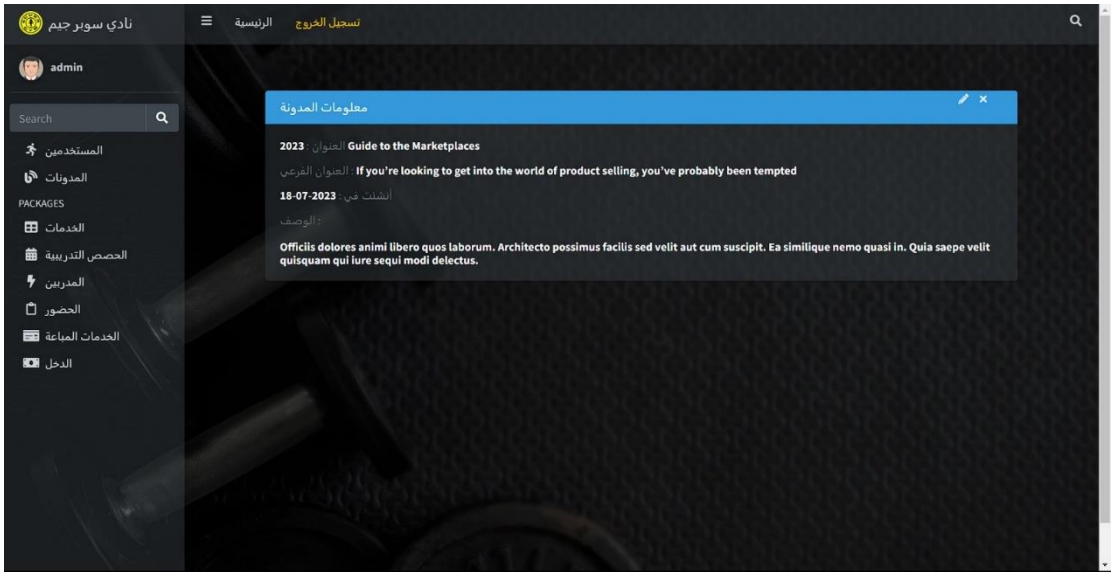


Figure 27: Dashboard - Admin - View in detailed blog data

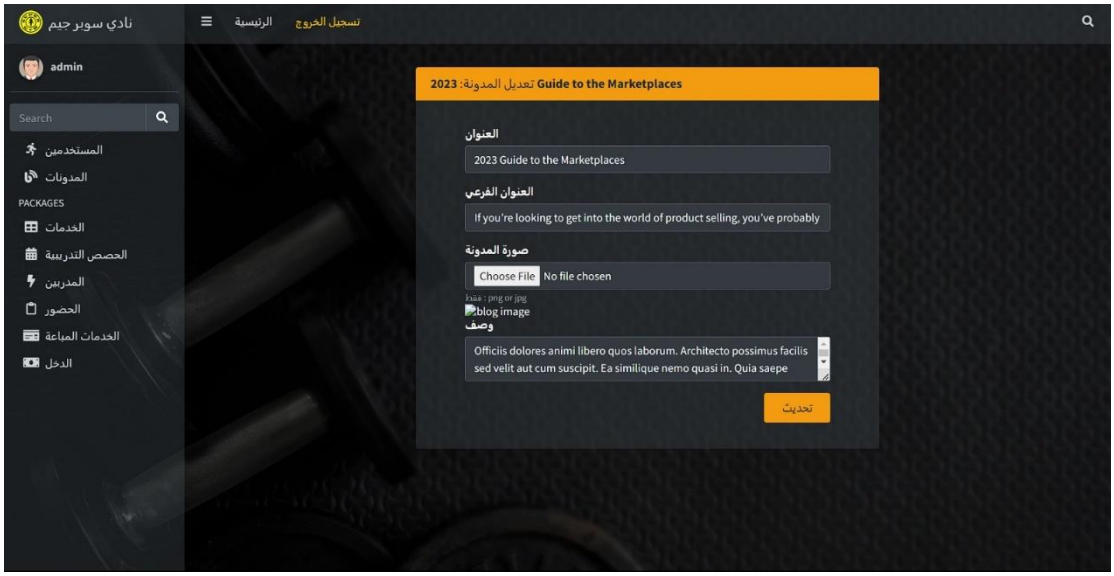


Figure 28: Dashboard - Admin - Update blog data

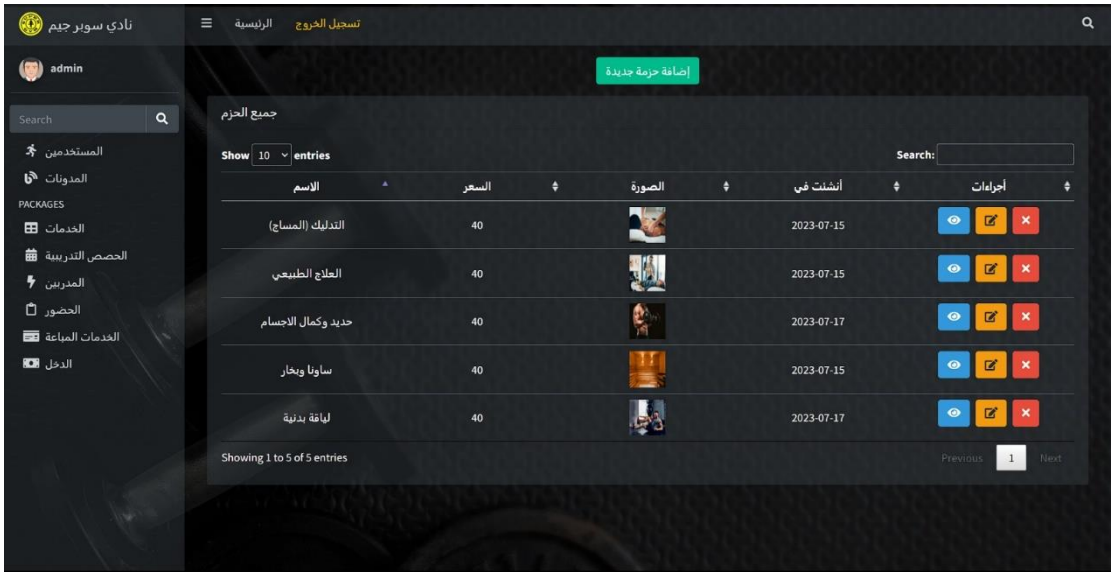


Figure 29: Dashboard - Admin - Services Table

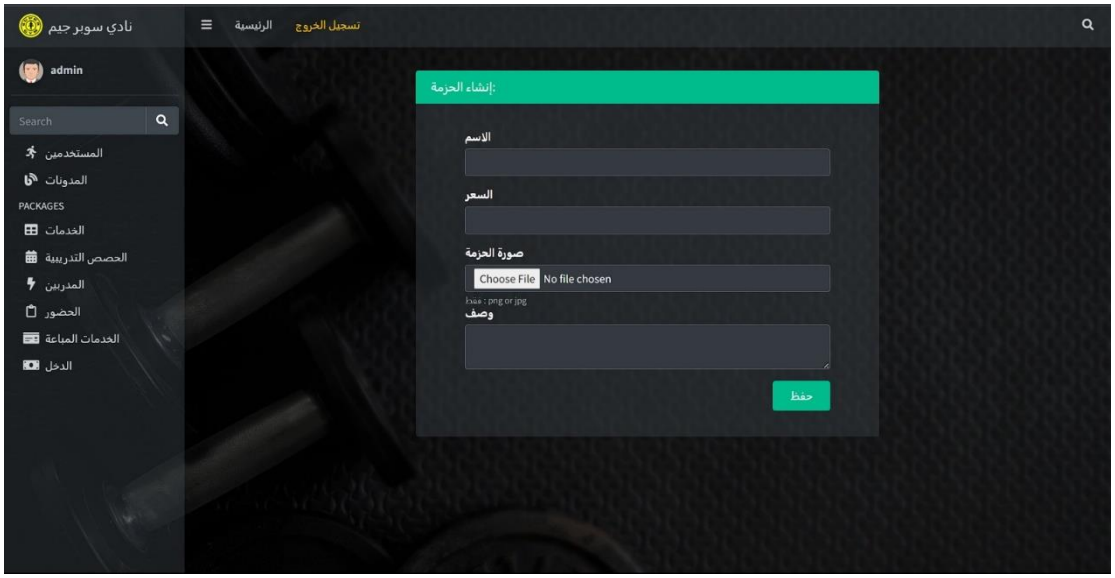


Figure 30: Dashboard - Admin - Insert a new service

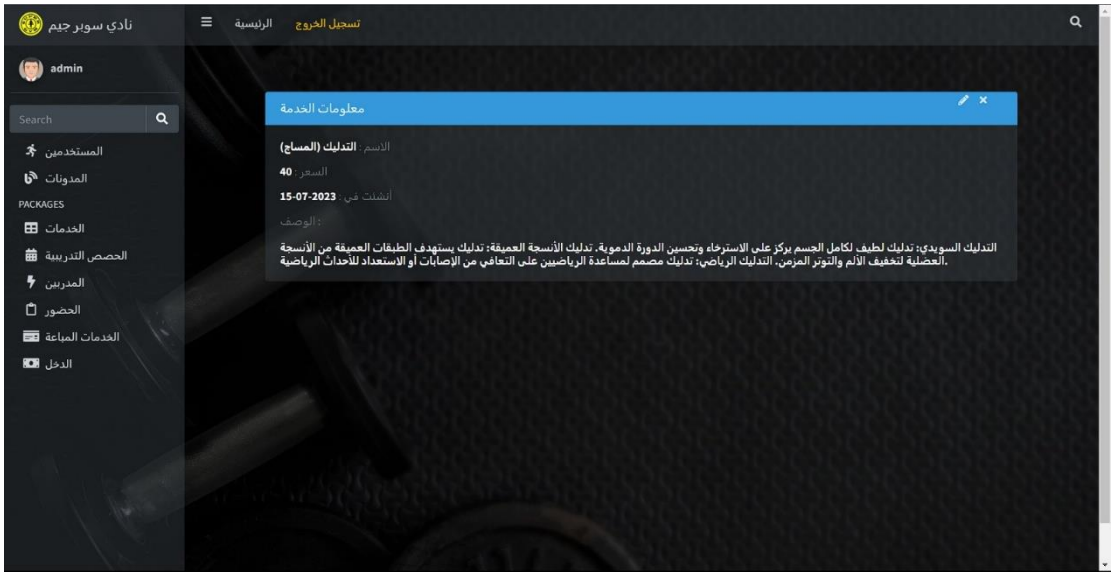


Figure 31: Dashboard - Admin - View in details service data

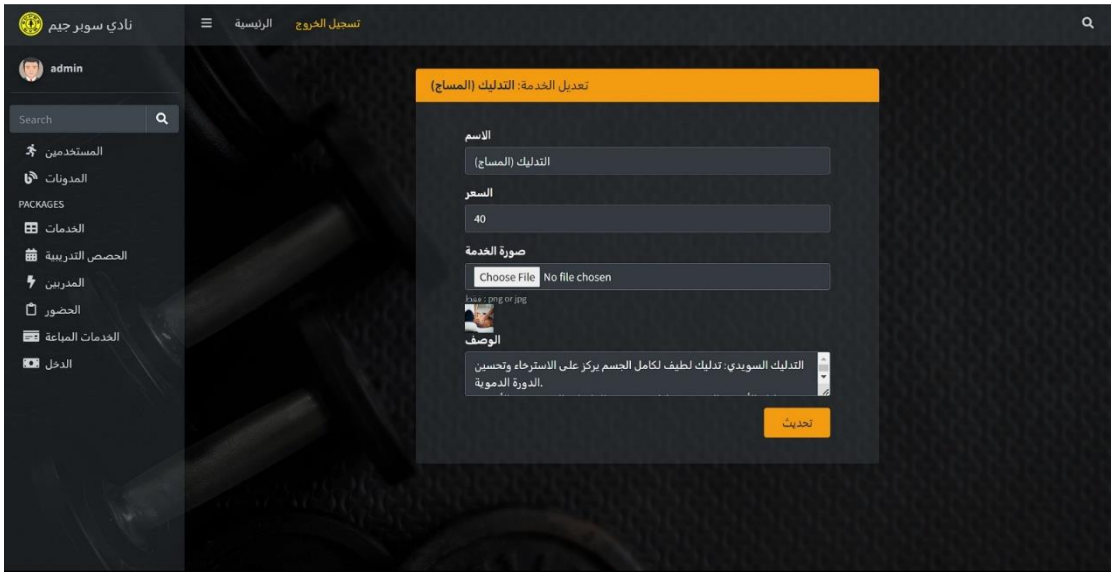


Figure 32: Dashboard - Admin - Update service data

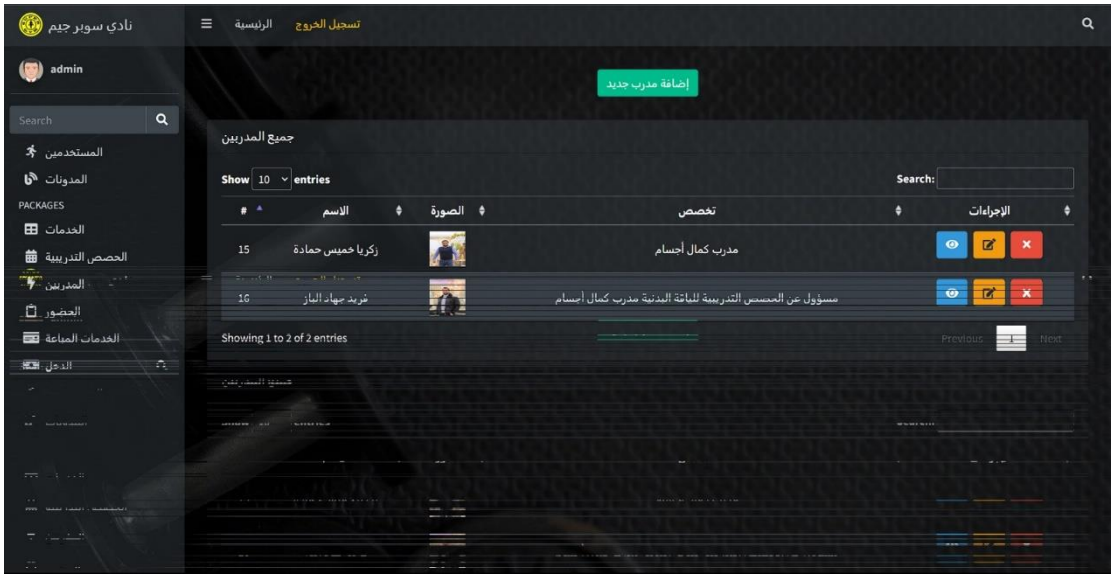


Figure 33: Dashboard - Admin - Trainer Table

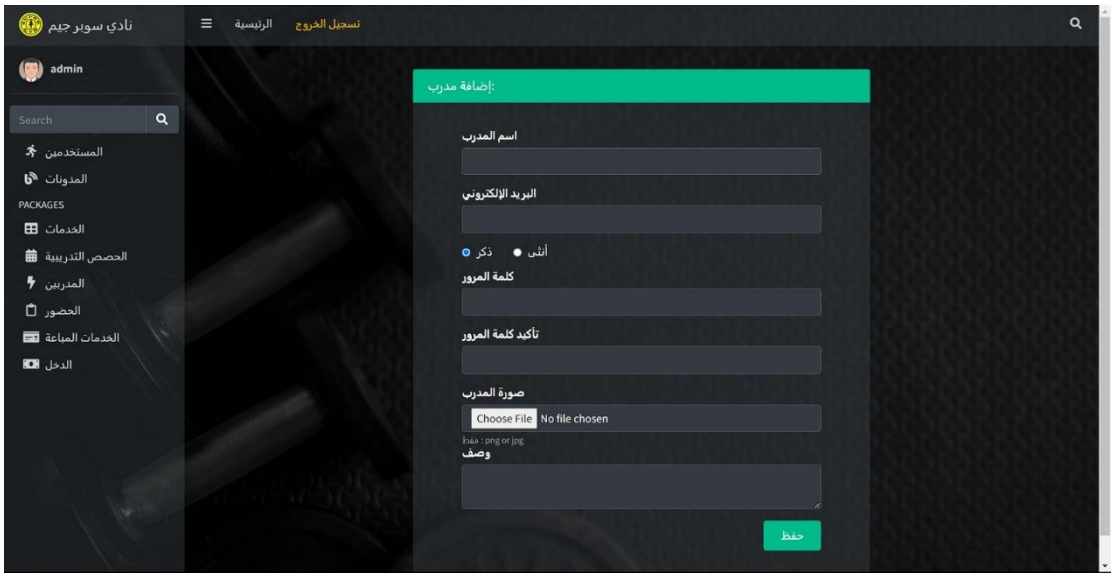


Figure 34: Dashboard - Admin - Insert a new trainer

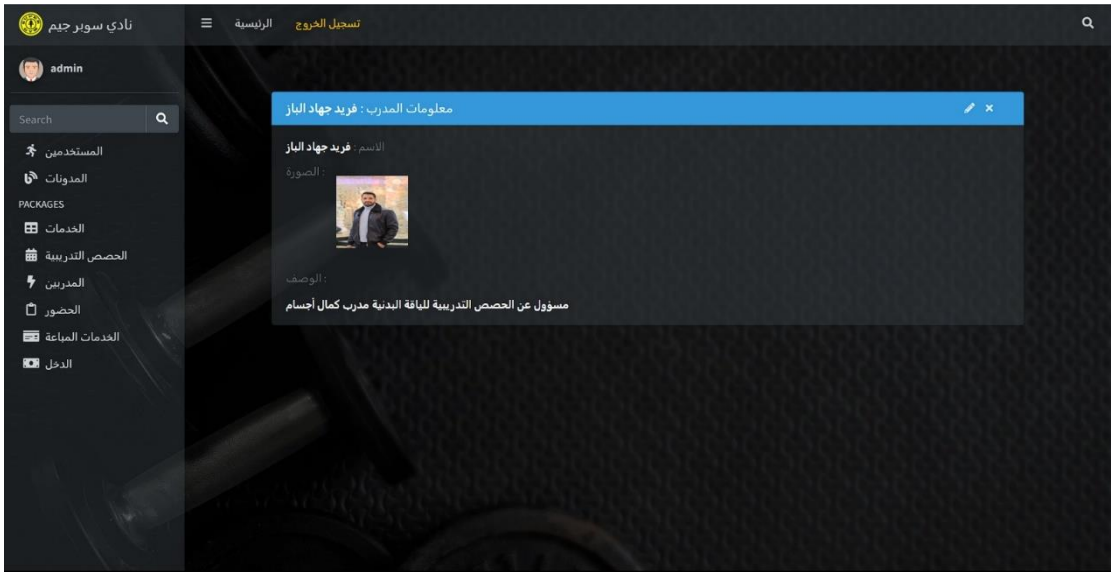


Figure 35: Dashboard - Admin - View in details trainer data

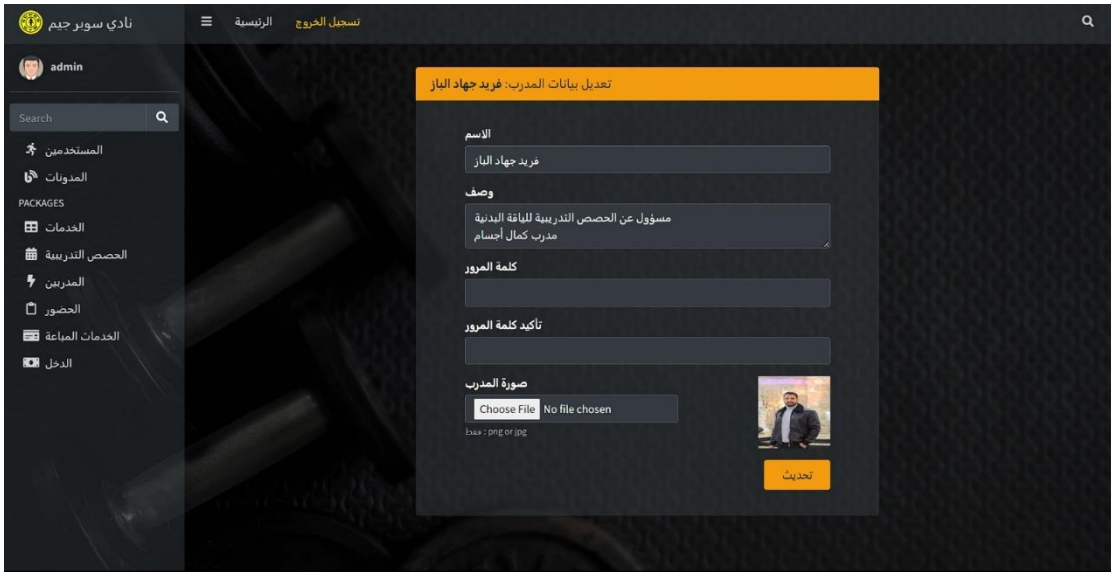


Figure 36: Dashboard - Admin - Update trainer data

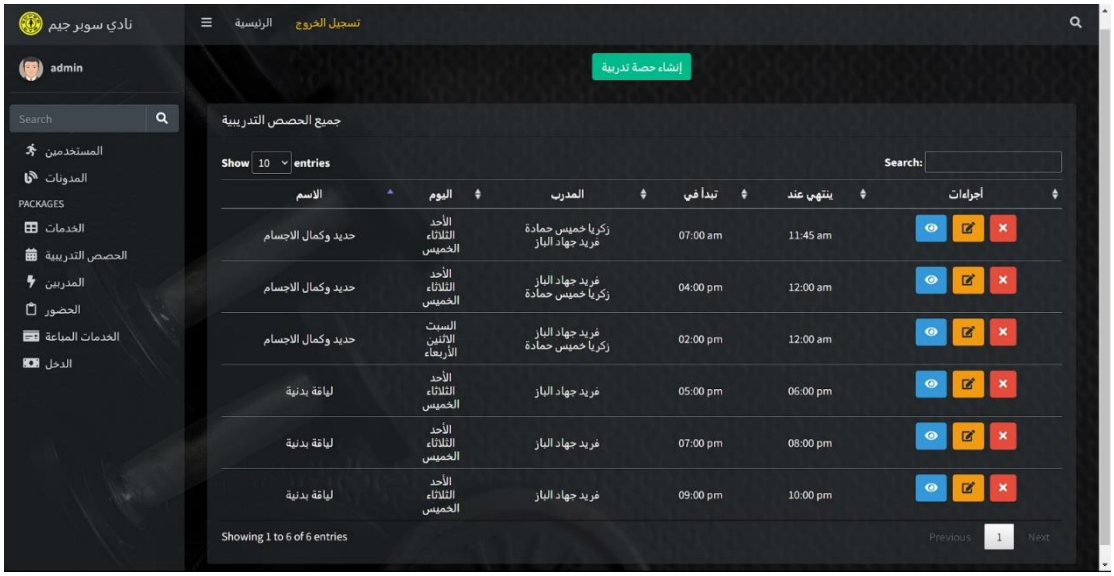


Figure 37: Dashboard - Admin - Trainning Session Table

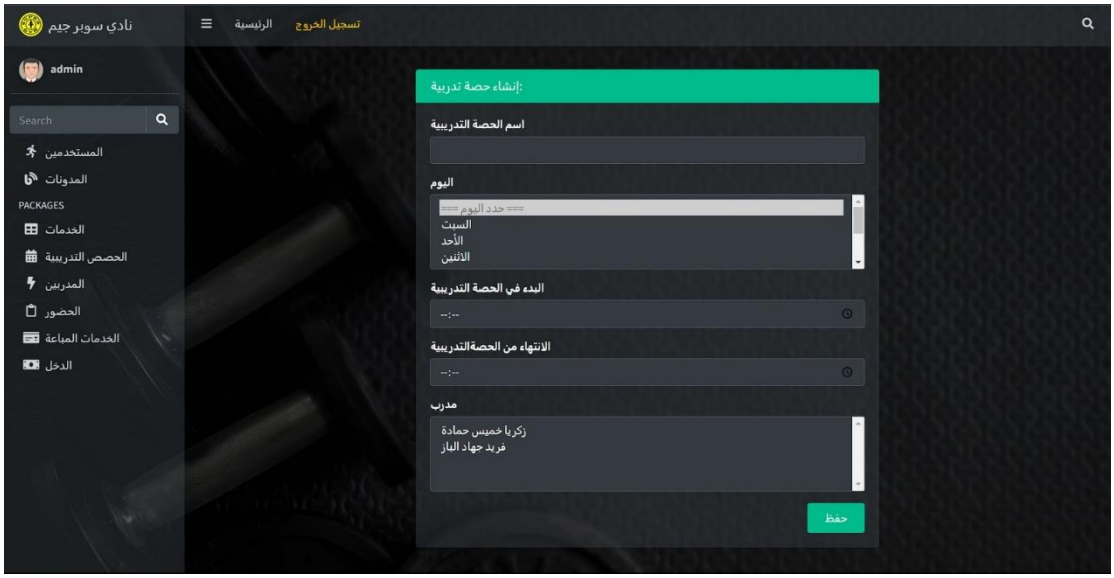


Figure 38: Dashboard - Admin - Insert a new session

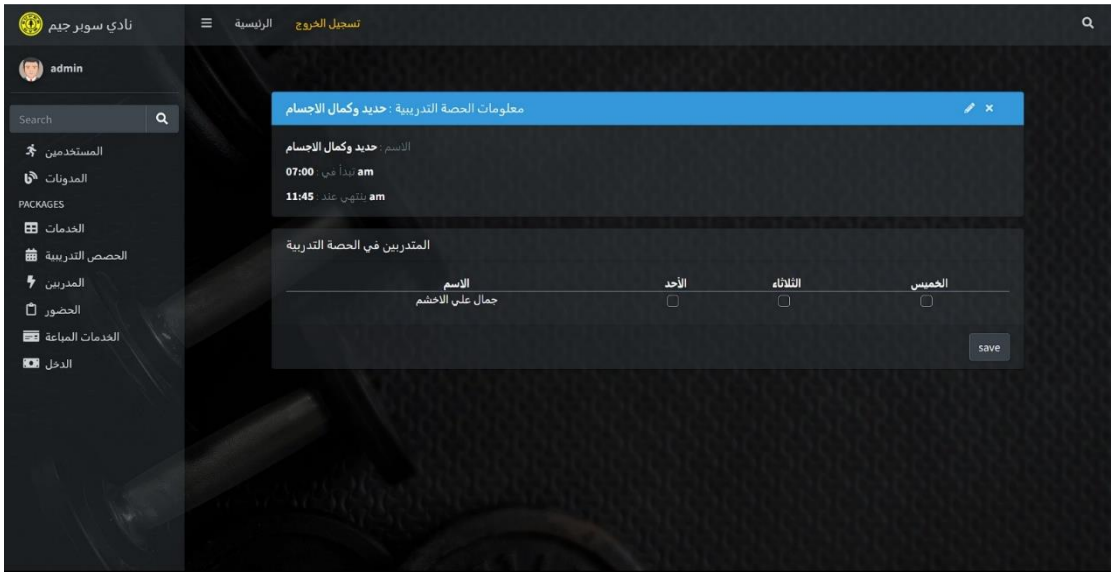


Figure 39: Dashboard - Admin - View in details session data

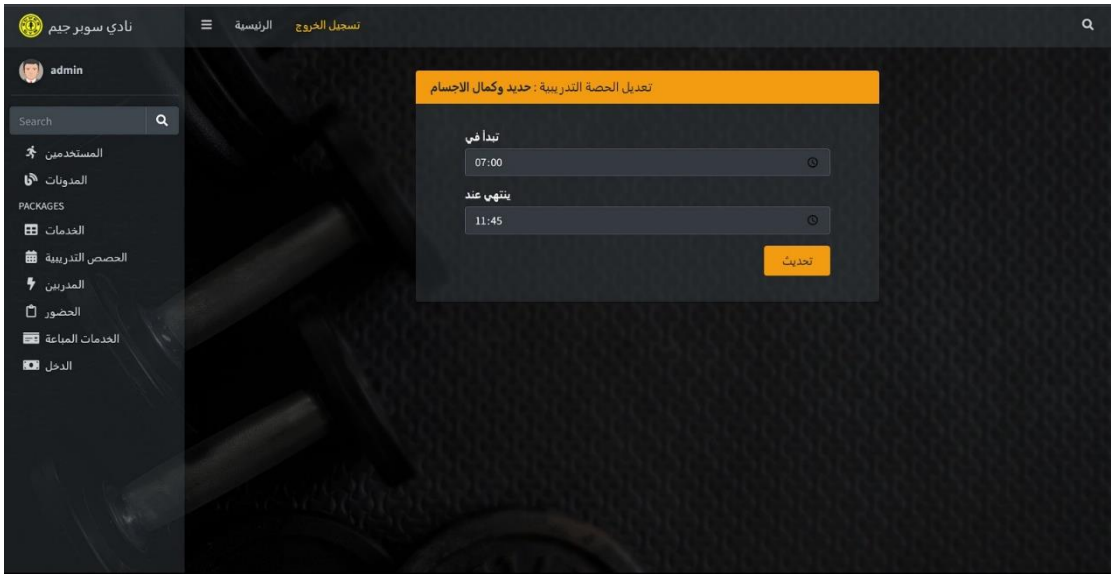


Figure 40: Dashboard - Admin - Update session time

Figure 41: Dashboard - Admin - Attendence Table

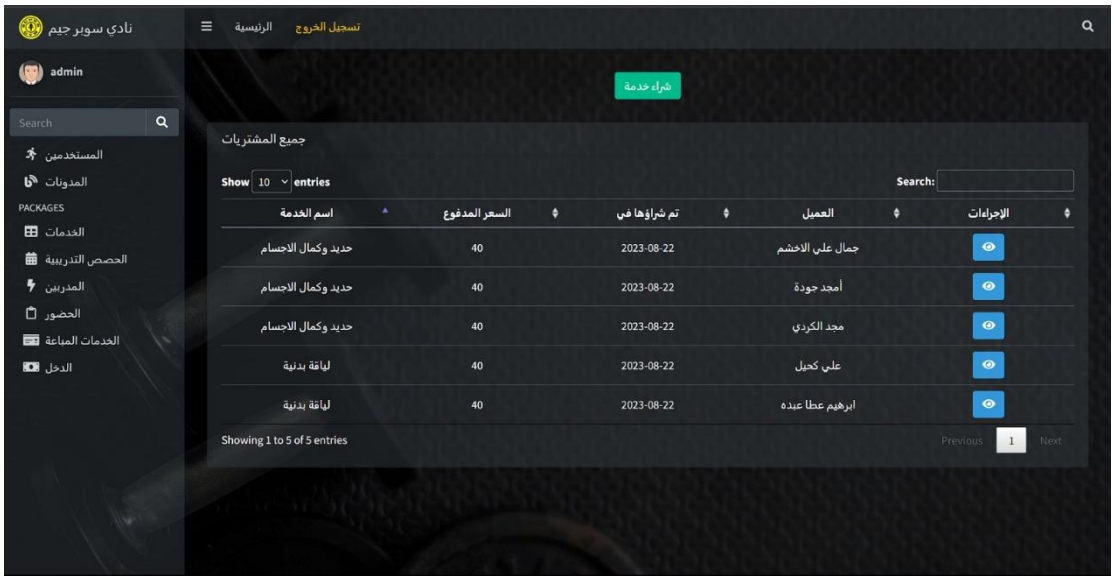


Figure 42: Dashboard - Admin - Purchased Services

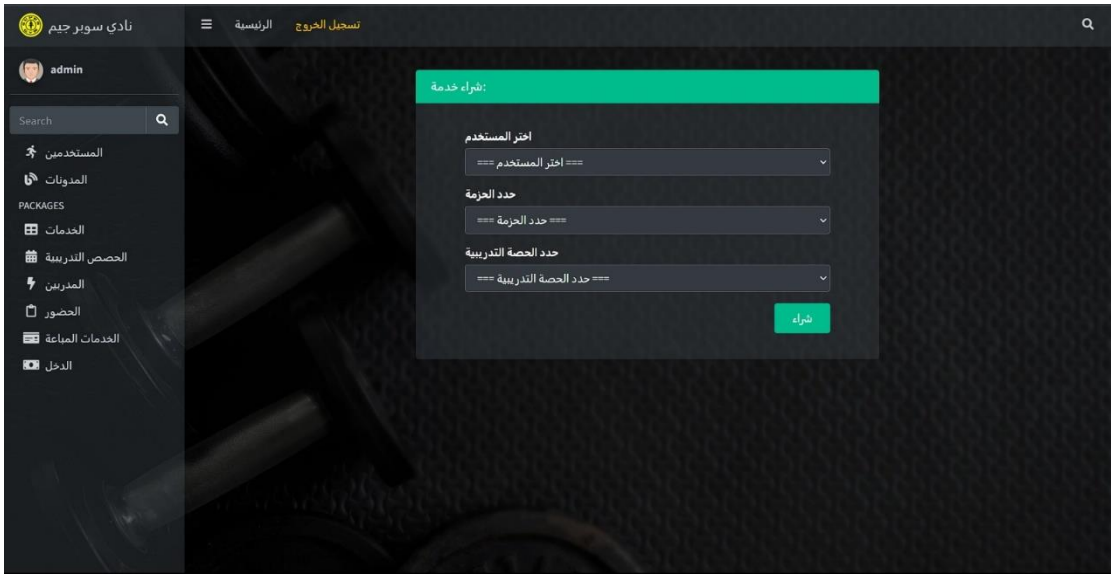


Figure 43: Dashboard - Admin - Purchase Service

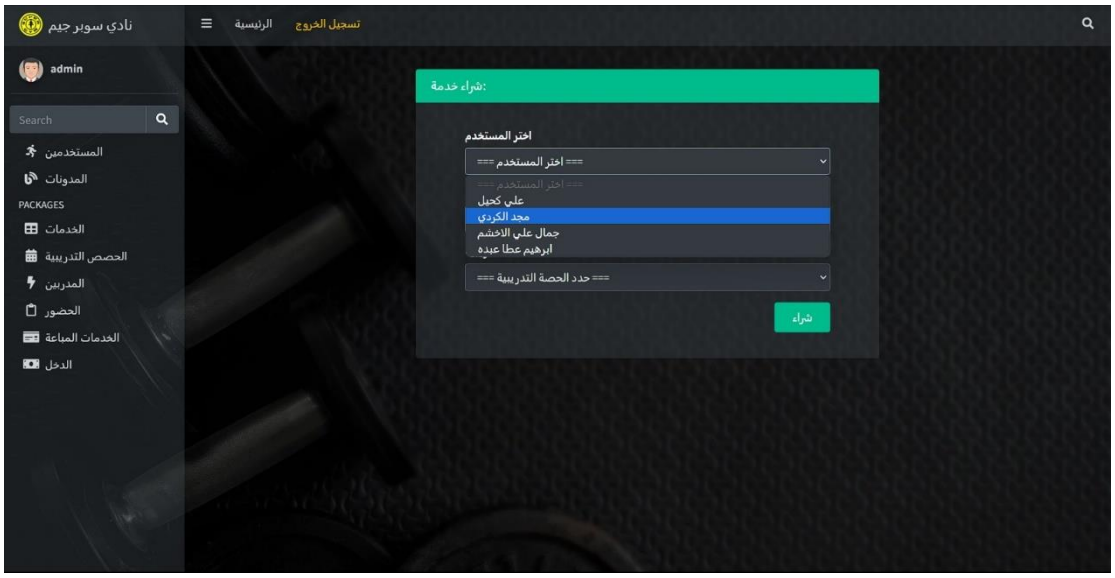


Figure 44: Dashboard - Admin - Members Options

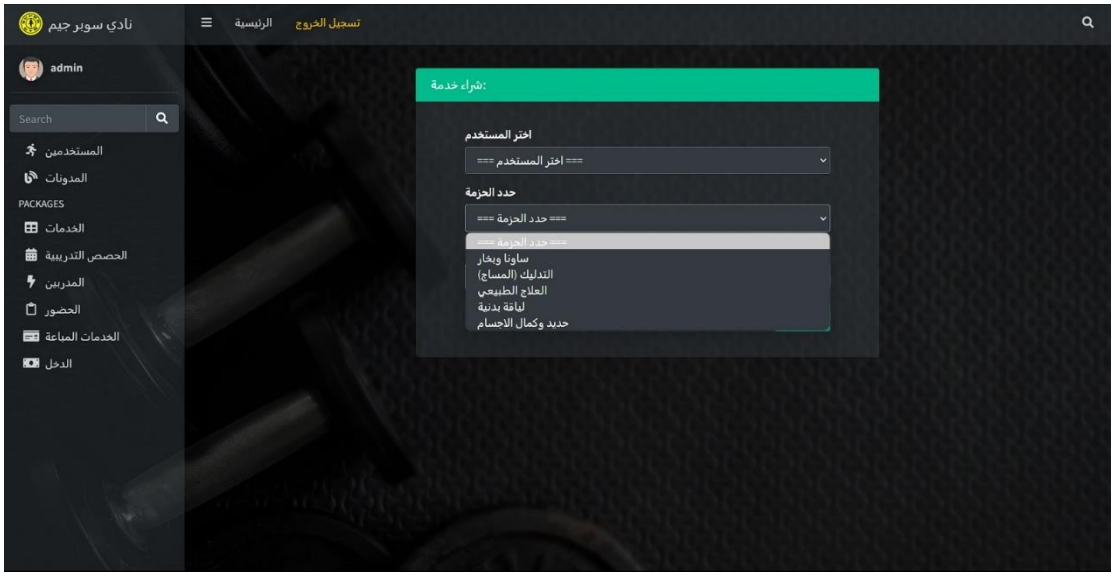


Figure 45: Dashboard - Admin - Services Option

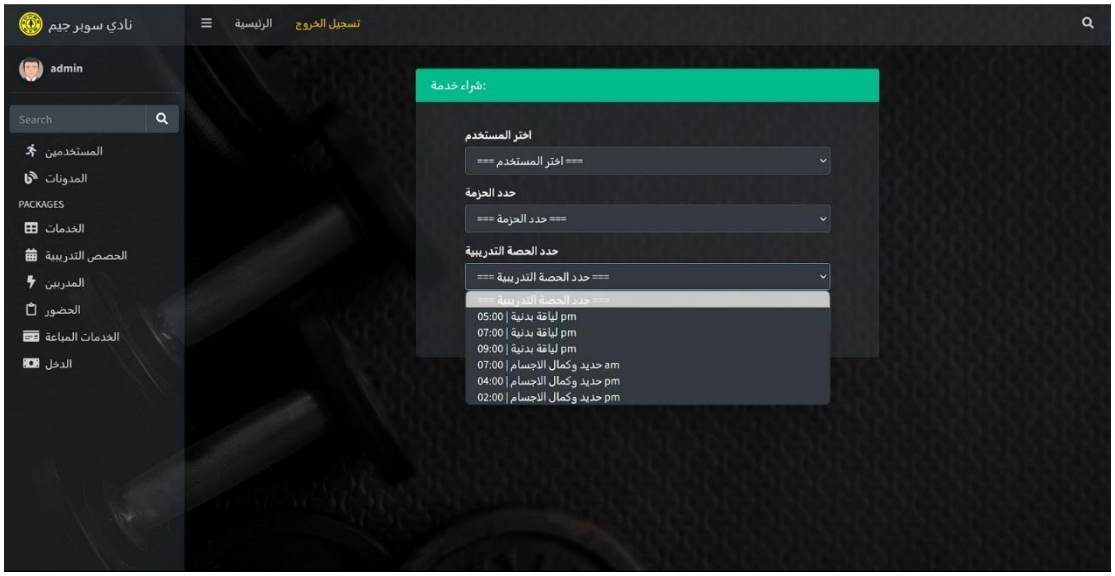


Figure 46: Dashboard - Admin - Session List

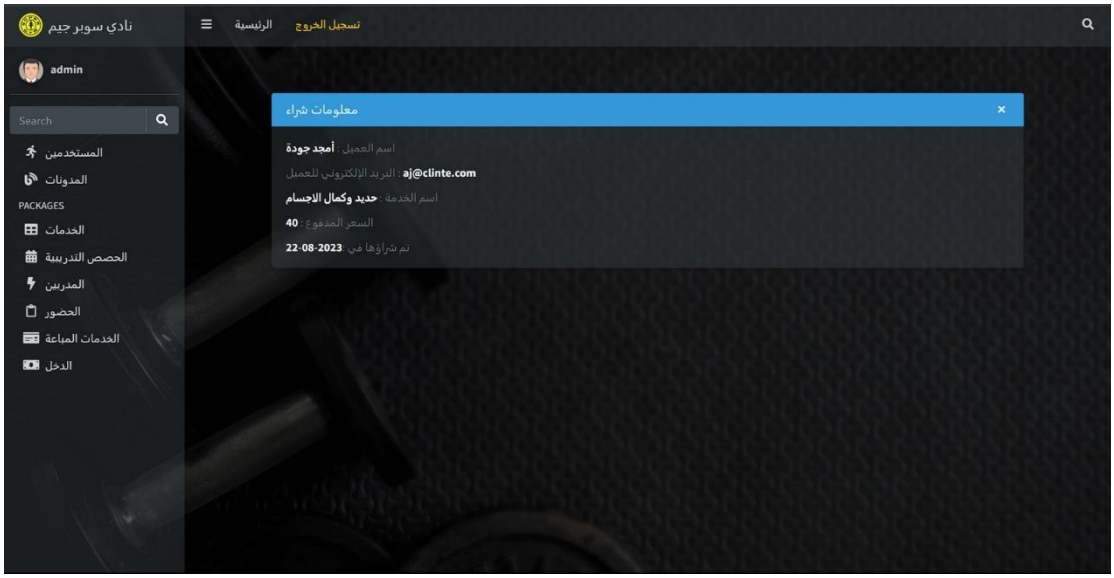


Figure 47: Dashboard - Admin - View in details purchase service

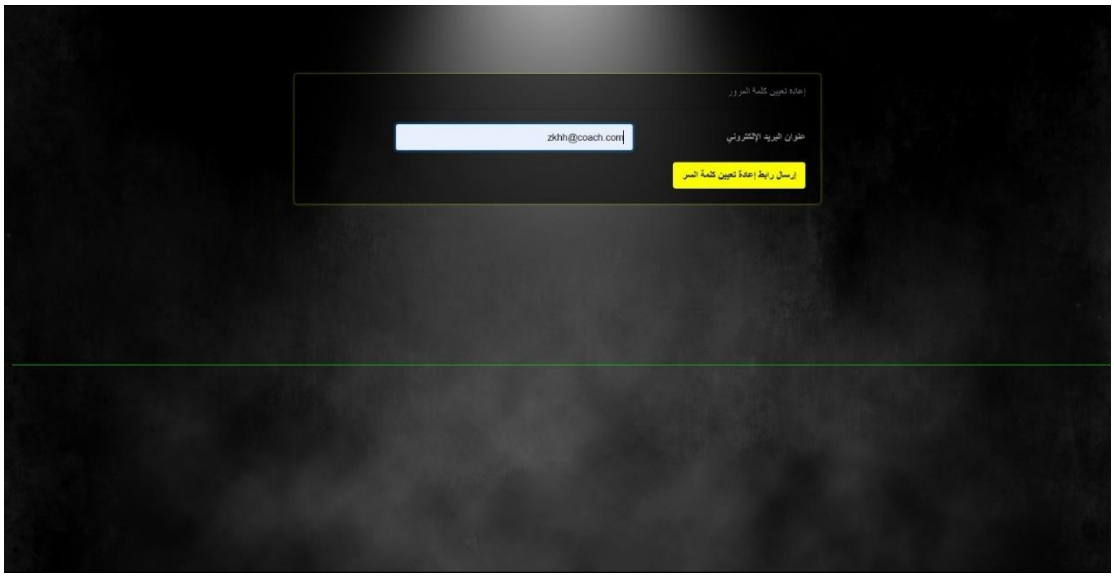


Figure 48: Dashboard - Admin - Reset password screen

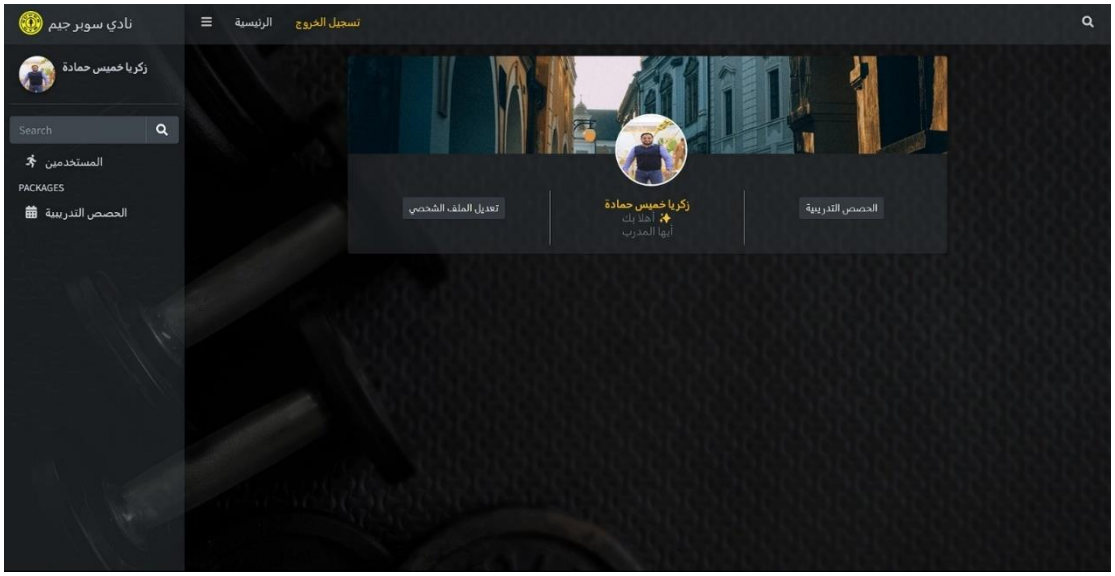


Figure 49: Dashboard - Trainer Screens

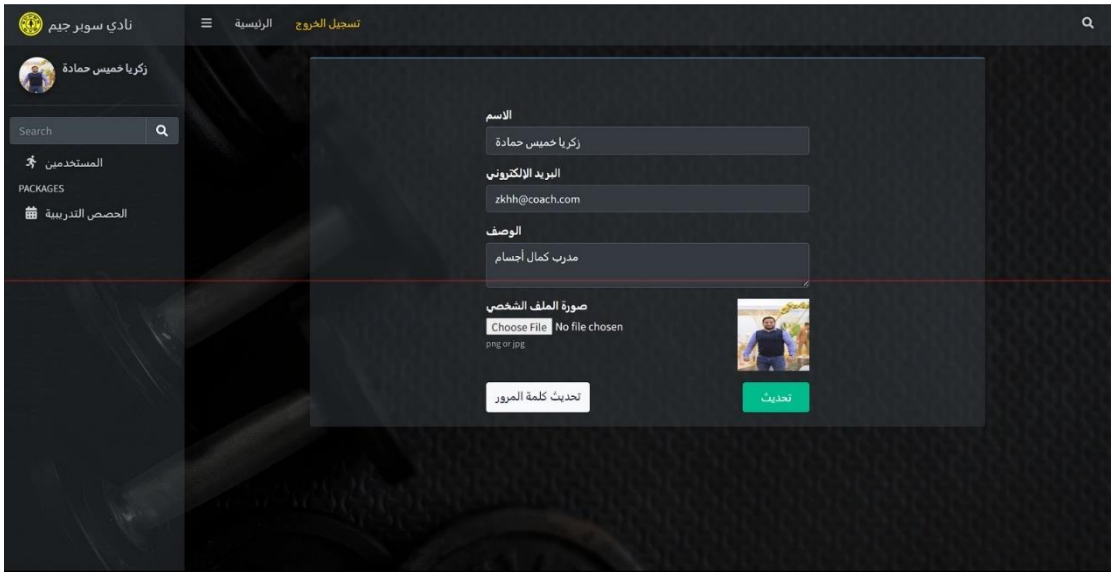


Figure 50: Dashboard - Trainer - Profile Update

Figure 51: Dashboard - Trainer - Coach trainees

Figure 52: Dashboard - Trainer - Coach Sessions

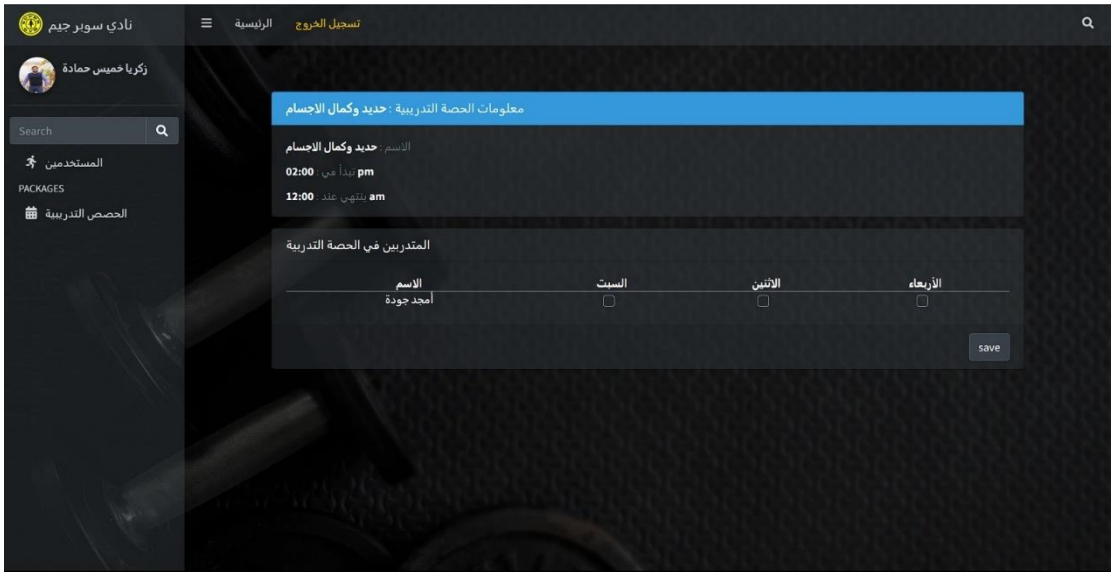


Figure 53: Dashboard - Trainer - Session informations

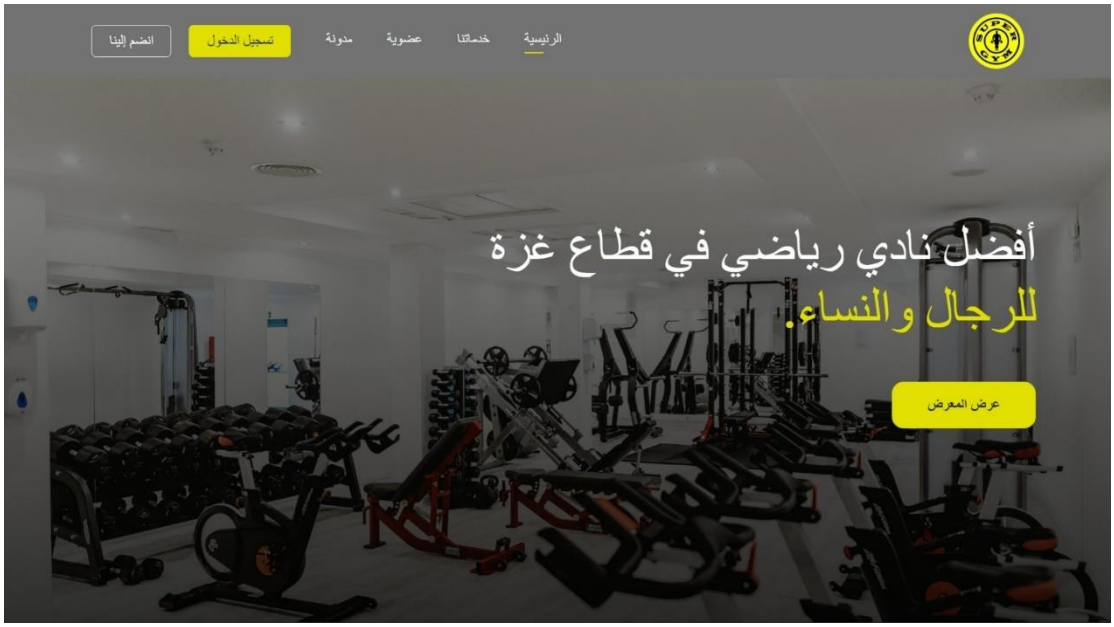


Figure 54: Home - Header Section

Figure 55: Home - About Us Section

Figure 56: Home - Our Services Section

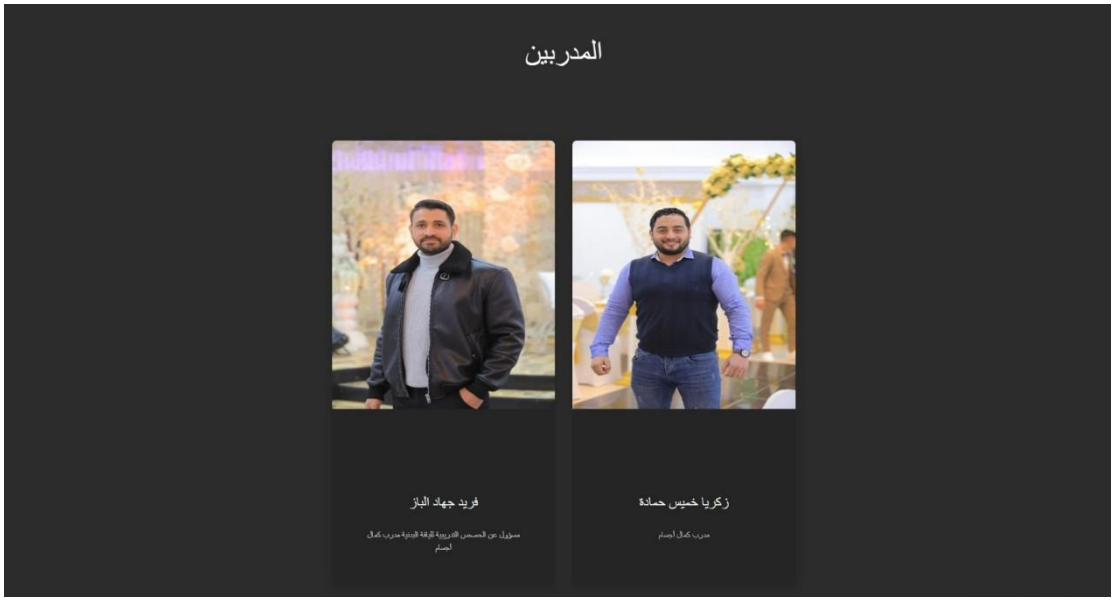


Figure 57: Home - Trainers Section

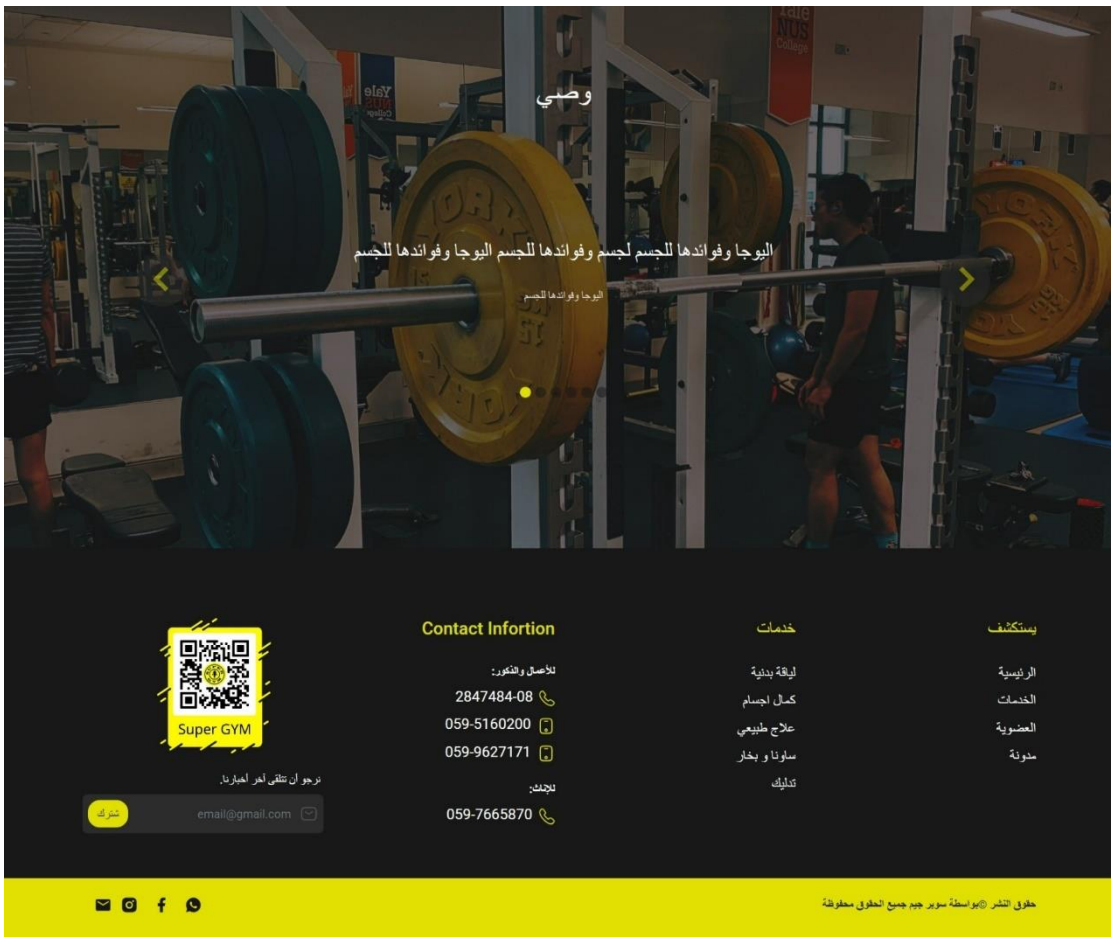


Figure 58: Home - Recommondation & Footer Section

Figure 59: Services Screen 01

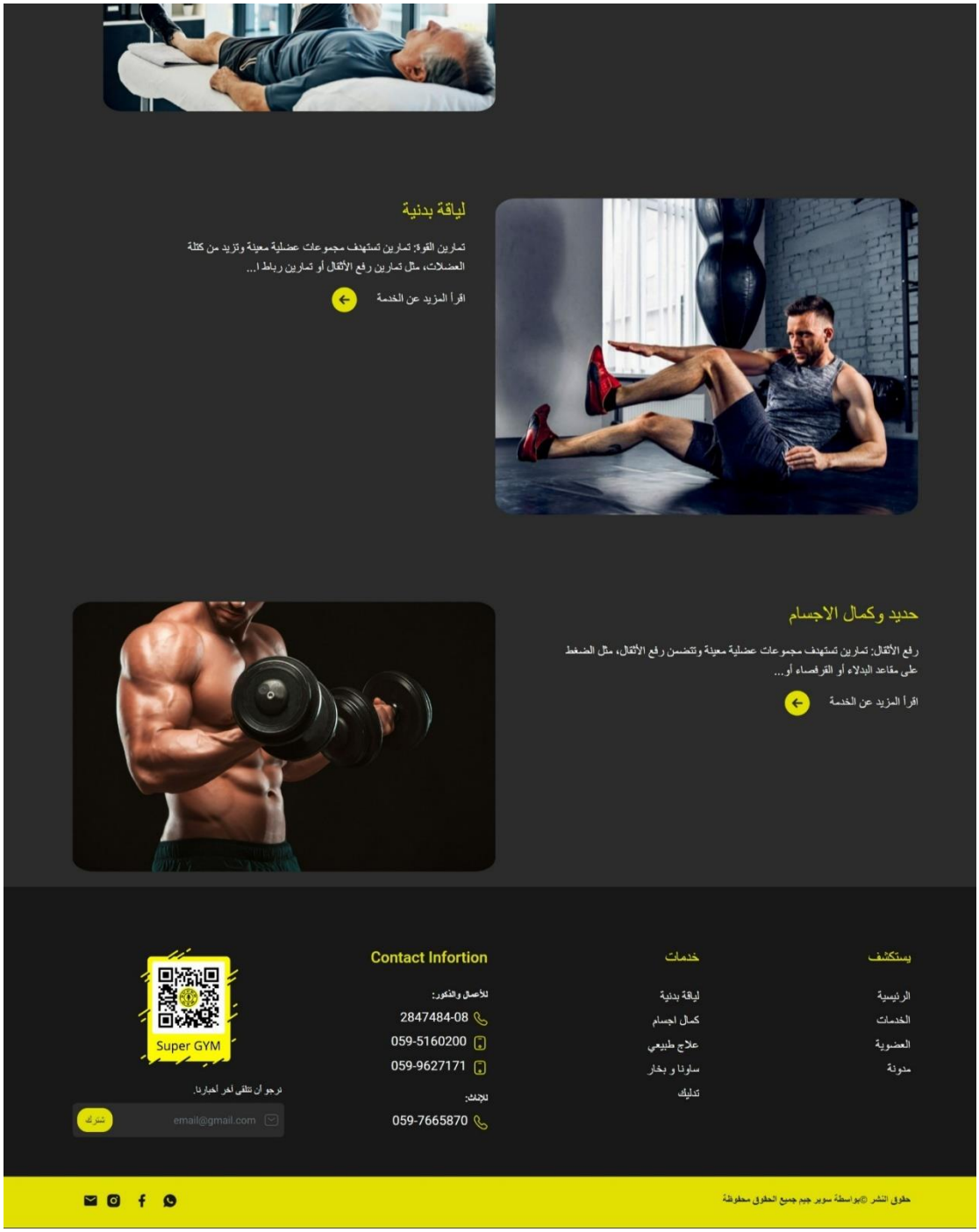


Figure 60: Services Screen 02

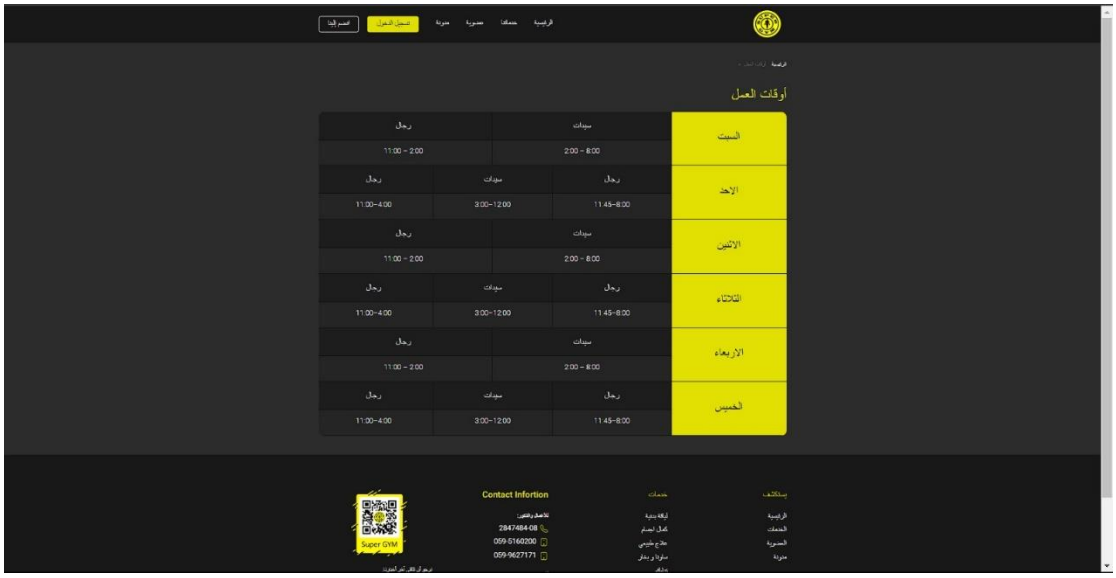


Figure 61: Membership - Working Time

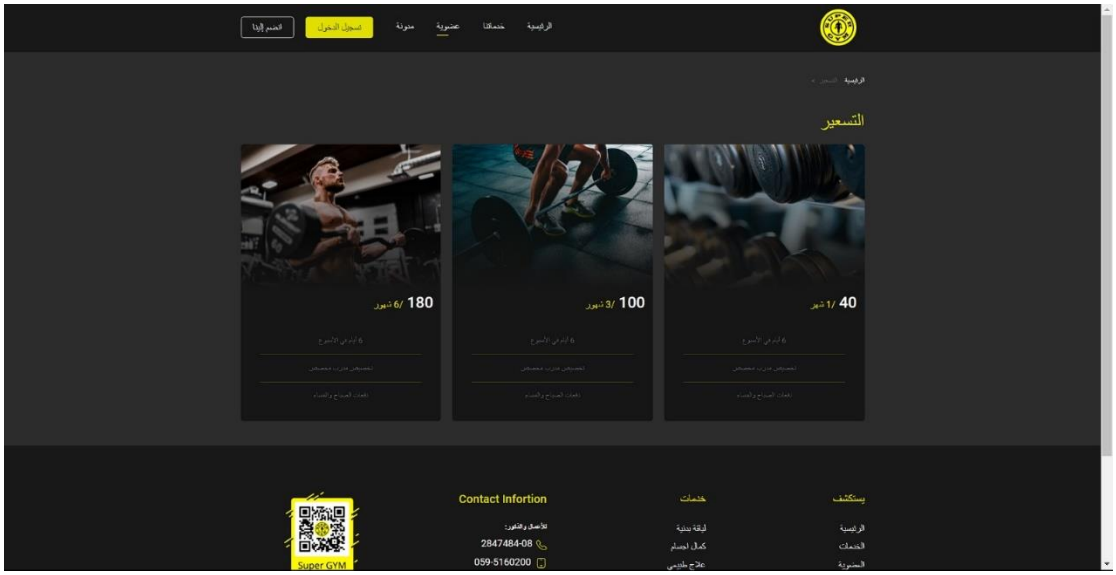


Figure 62: Membership - Plans

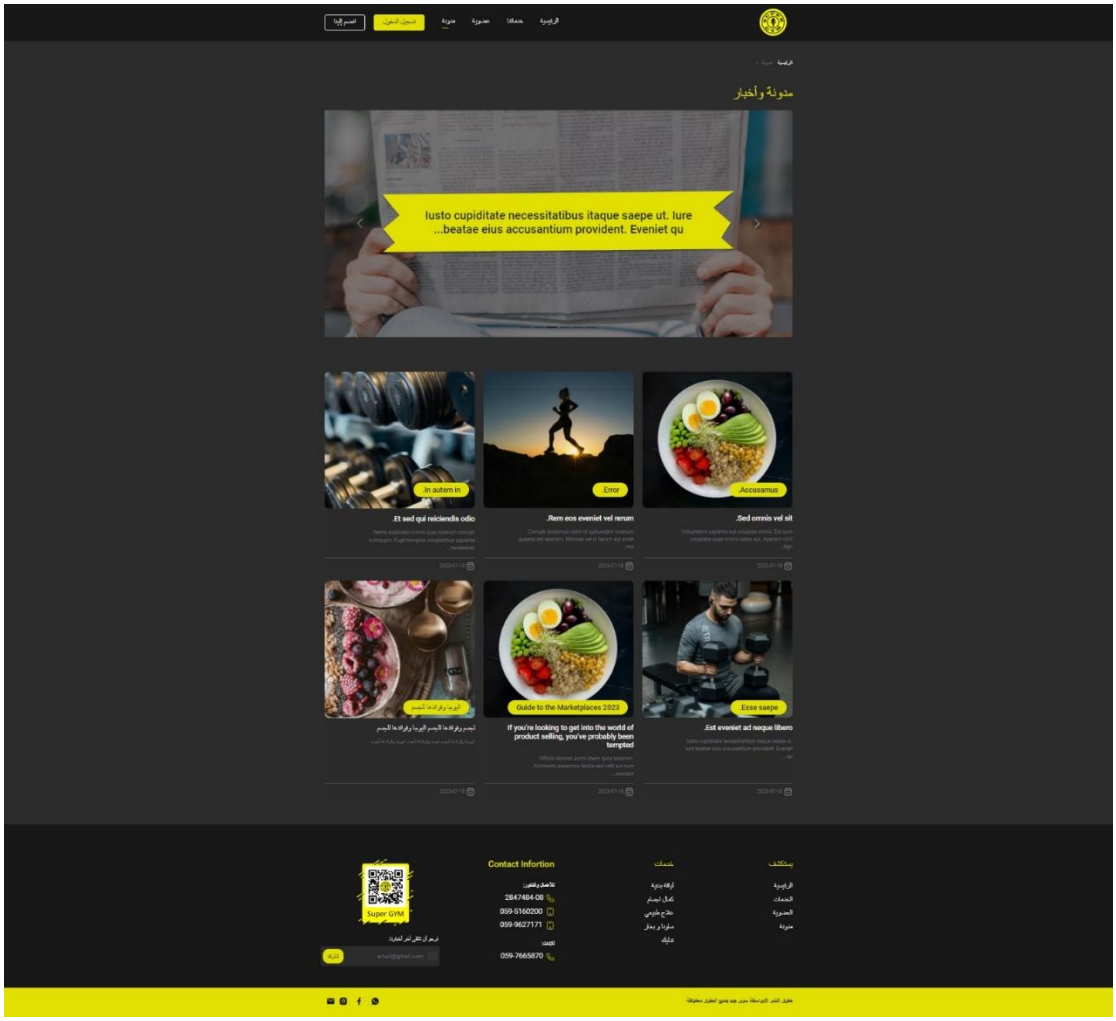


Figure 63: Blogs Screen

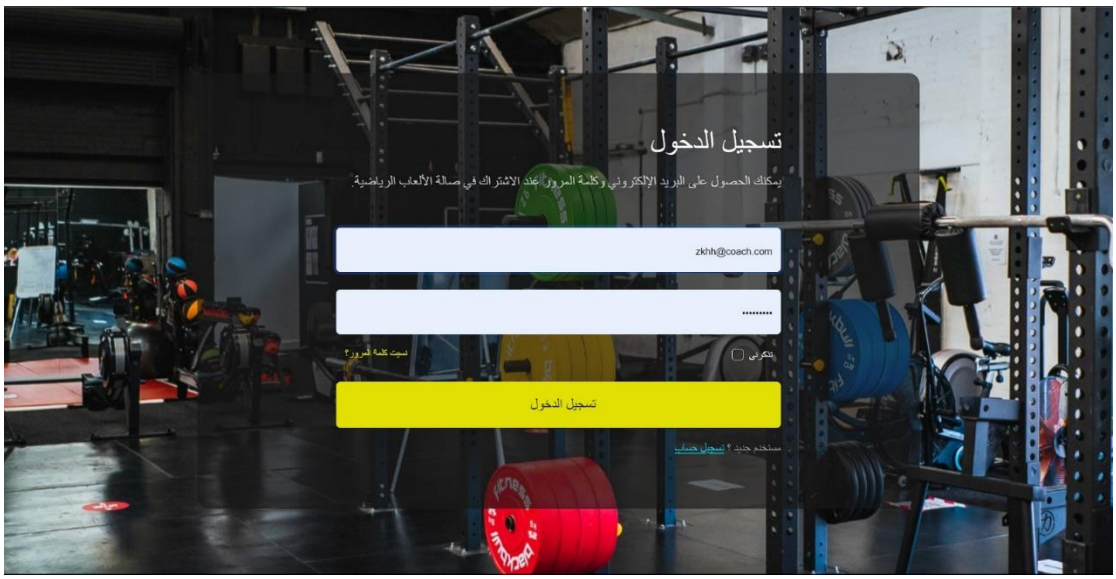


Figure 64: Login Screen

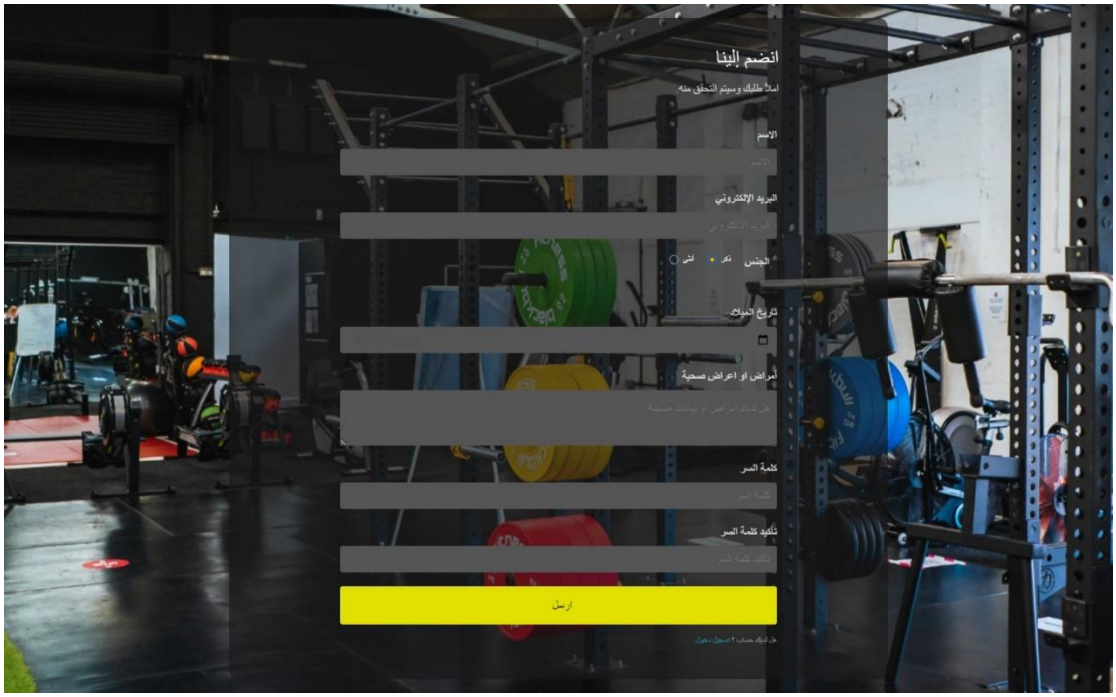


Figure 65: Sign up Screen

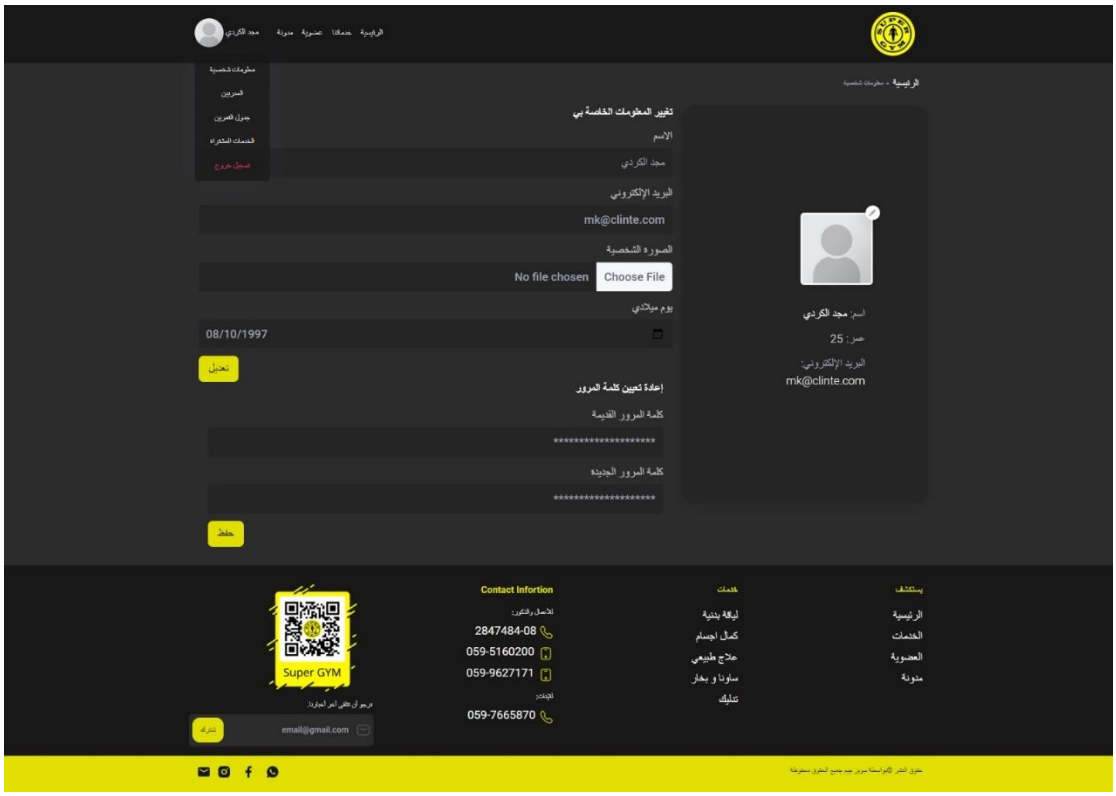


Figure 66: Update Profile Screen

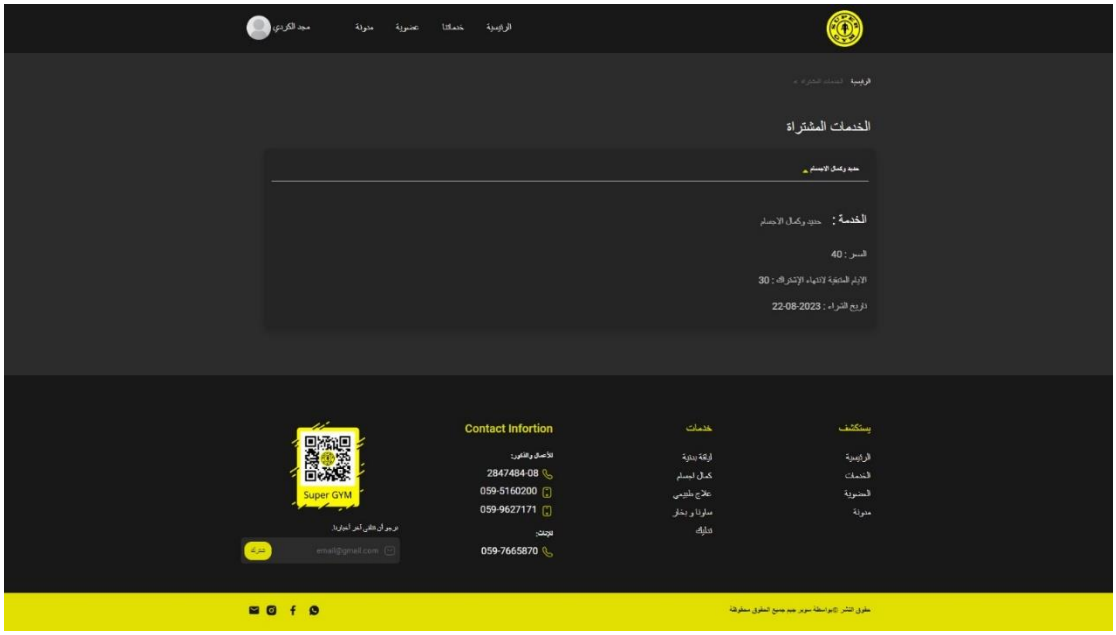


Figure 67: Purchased Services Screen

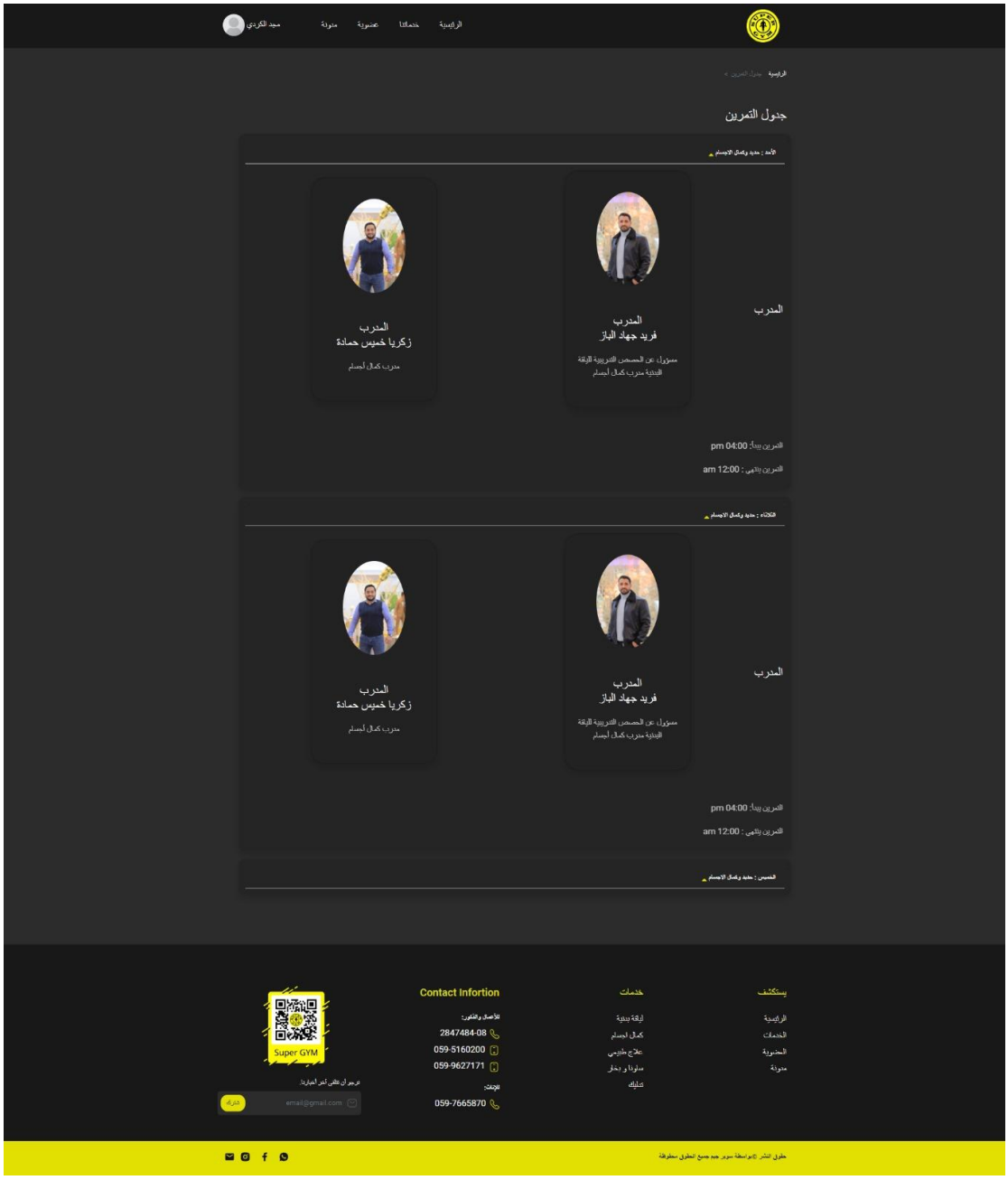


Figure 68: Session schedule

Building the System and Testing

Project Management & Planning

In this section we will review the actual teamwork of the team and the roles played by each one of them to accomplish this project. Here is a table shows the project milestones and members who accomplished it.

Our Team consists of 4 members: *Belal Yasin, Mohammed Riyad, Fadi Mohammed, and Amjad Adham.*

Table 2: Project Milestones & Actual Teamwork

Project Milestones	Members Worked On
Data Collection	*The Whole Team*
Analysis Requirements	
Design	
System Structure & Database ERD	*Bela Yasin & Amjad Adham*
UIUX Design	*Fadi Mohammed, Mohammed Riyad, & Amjad Adham*
Development	
Database Development	*Bela Yasin & Amjad Adham*
Back-End & Business Logic Implementation	*Belal Yasin*
Front-End Design Implementation	*Fadi Mohammed, Mohammed Riyad, & Amjad Adham*
Testing	
Back-End Testing	*Bela Yasin & Amjad Adham*
Front-End Testing	*Fadi Mohammed, Mohammed Riyad*

Launching / Production Step	
Upload The Site to Hosting	*Belal Yasin*
Documenting The Project	*The Whole Team*

Main Functions Code Snippets

Here is a code snippet with description form main functionalities:

Gym-Managment-System - UserController.php

```php

/**
 * This function is responsible for handling the creation (registration) of a new user.
 */
public function store(StoreUserRequest $request)
{
    $validator = Validator::make($request->all(), [
        'name' => 'required|string|min:3',
        'email' => 'required|unique:users|email',
        'password' => 'required|min:6|max:20',
        'confirmPassword' => 'required|same:password',
        'date_of_birth' => 'required|date',
    ]);
    if ($validator->fails()) {
        return Redirect::back()->withErrors($validator->errors());
    }
    $img = $request->file('profile_img');

    if ($img != null) :
        $imageName = time() . rand(1, 200) . '.' . $img->extension();
        $img->move(public_path('imgs//' . 'users'), $imageName);
    else :
        $imageName = 'Client.Png';
    endif;

    // handle creator
    $newUser = new User();
    $newUser->name = $request->input('name');
    $newUser->email = $request->input('email');
    $newUser->password = Hash::make($request->input('password'));
    $newUser->profile_img = $imageName;
    $newUser->date_of_birth = $request->input('date_of_birth');
    $newUser->gender = $request->input('gender');
    $newUser->description = $request->input('description');
    $newUser->email_verified_at = now();
    $newUser->assignRole('client');
    $newUser->save();
    //redirection to posts.index
    return redirect()->route('users.index');
}
```

Figure 69: Code Snippet - User Controller - Create User

Gym-Managment-System - UserController.php

```php
/**
     * This function is responsible for displaying the details of a user based on their user ID.
     */
    public function show($userID)
    {
        $user = User::findOrFail($userID);
        return view('users.show', ['user' => $user]);
    }
```

Figure 70: Code Snippet - User Controller - Show Single User

Gym-Managment-System - UserController.php

```php
/**
     * This function handles the update of an existing user's information.
     */
    public function update(Request $request, User $user)
    {
        $validator = Validator::make($request->all(), [
            'name' => 'required|string|min:3',
            'email' => 'required|email|unique:users,email,' . $user->id,
            'date_of_birth' => 'required|date',
        ]);
        if ($validator->fails()) {
            return Redirect::back()->withErrors($validator->errors());
        }
        $user->name = $request->input('name');
        $user->email = $request->input('email');
        $user->date_of_birth = $request->input('date_of_birth');
        $isSaved = $user->save();

        return redirect()->route('users.index');
    }
```

Figure 71: Code Snippet - User Controller - Update User

Gym-Managment-System - UserController.php

```php
/**
     * This function handles the update of a user's profile information.
     */
    public function updateProfile(Request $request)
    {
        $userID = $request->id;

        $validated = $request->validate([
            'name' => 'required|max:50|min:3',
            'email' => ['required', 'string', 'email', 'max:255', 'unique:users,email,' . $userID],
            'date_of_birth' => 'required|date',
        ]);

        $oldimg = $request->oldimg;

        if ($request->hasFile('profile_img')) {
            $request->validate([
                'profile_img' => 'image | mimes:jpg,jpeg,png',
            ]);

            $imageName = time() . '.' . $request->file('profile_img')->extension();
            $request->file('profile_img')->move(public_path('imgs//' . 'users'), $imageName);
            DB::table('users')->where('id', '=', $userID)->update(['profile_img' => $imageName]);

            if ($oldimg != "Client.png") {
                // to delete old image
                if (file::exists(public_path('imgs//' . 'users/' . $oldimg))) {
                    file::delete(public_path('imgs//' . 'users/' . $oldimg));
                }
            }
        }

        User::find($userID)->update([
            'name' => $validated['name'],
            'email' => $validated['email'],
            'date_of_birth' => $validated['date_of_birth'],
            //            'profile_img' => $imageName
        ]);
        return redirect()->route('dashboard');
    }
```

Figure 72: Code Snippet - User Controller - Update User Profile

Gym-Managment-System - UserController.php

```php
/**
     * This function handles the process of updating a user's password.
     */
    public function updatePassword(Request $request)
    {
        $userid = Auth::id();

        $data = $request->validate([
            'newpassword' => 'required|min:6',
            'oldpassword' => 'required|min:6'
        ]);

        $userPassword = Auth::user()->password;

        $newPassword = Hash::make($data['newpassword']);

        if (Hash::check($data['oldpassword'], $userPassword)) {     // check if enter old password correct or not

            DB::table('users')->where('id', '=', $userid)->update(['password' => $newPassword]);
            return redirect()->route('dashboard');
        } else {
            $msg = 'please enter your old password correctly';
            return view('profile.editPassword', ['msg' => $msg]);
        }
    }

```

Figure 73: Code Snippet - User Controller - Update User Password

Gym-Managment-System - UserController.php

```php
/**
      *  This function is responsible for banning a user.
      */
     public function ban($user)
     {
         User::findOrFail($user)->ban([
             'comment' => 'لقد تم انتهاء الإشتراك مع فترة التمديد!',
         ]);
         $verfiyUser = User::findOrFail($user);
         $verfiyUser->email_verified_at = null;
         $verfiyUser->save();
         return redirect()->route('users.index');
     }
```

Figure 74: Code Snippet - User Controller - Ban User

Gym-Managment-System - UserController.php

```php
/**
     * This function is responsible for unbanning a user.
     */
    public function unban($user)
    {
        User::findOrFail($user)->unban();
        $verfiyUser = User::findOrFail($user);
        $verfiyUser->email_verified_at = now();
        $verfiyUser->save();
        return redirect()->route('users.index');
    }
```

Figure 75: Code Snippet - User Controller - Unban User

Gym-Managment-System - UserController.php

```php
/**
     * This function is responsible for rendering the index view for users,
     * considering different scenarios based on the user's role and authentication guard.
     */
    public function index()
    {
        $isWeb = auth()->guard('web')->check();
        if ($isWeb) {
            $roleAdmin = auth()->user()->hasRole('admin');
            $gender = auth()->user()->gender;
            if ($roleAdmin) {
                if ($gender === 'male') {
                    $users = User::role('client')->where('gender', 'male')->get();
                } elseif ($gender === 'female') {
                    $users = User::role('client')->where('gender', 'female')->get();
                }
            } else {
                $users = User::role('client')->get();
            }
        } else {
            $isCoach = auth()->guard('coach')->check();
            $roleCoach = auth('coach')->user()->hasRole('coach');
            $gender = auth('coach')->user()->gender;
            if ($roleCoach) {
                $coach = auth('coach')->user();
                $trainingSessions = $coach->trainingSessions;
                $attendances = Attendance::whereIn('training_session_id', $trainingSessions->pluck('id'))->get();
                $usersInSessions = $trainingSessions->pluck('users')->collapse()->filter(function ($user) use ($gender) {
                    return $user->gender === $gender;
                })->unique();
                $users = $usersInSessions;
            }
            return view('users.index', data: [
                'users' => $users,
                'attendances' => $attendances
            ]);
        }

        return view('users.index', data: [
            'users' => $users,
        ]);
    }
```

Figure 76: Code Snippet - User Controller - View All Users

Gym-Managment-System - UserController.php

```php
/**
     * This function is responsible for deleting a user's account.
     */
    public function destroy($userId)
    {
        $checkAttendance = Attendance::where('user_id', $userId)->first();
        $checkBuyPackage = BuyPackage::where('user_id', $userId)->first();


        if ($checkAttendance == null && $checkBuyPackage == null) {

            $oldimg = User::where('id', $userId)->first()->profile_img;
            if ($oldimg != "Client.png") {
                // to delete old image
                if (file::exists(public_path('imgs//' . 'users/' . $oldimg))) {
                    file::delete(public_path('imgs//' . 'users/' . $oldimg));
                }
            }

            User::findOrFail($userId)->delete();
            return to_route('users.index')
                ->with('success', 'user deleted successfully');
        } else {
            return redirect()->route('users.index')
                ->with('errorMessage', 'cannt be deleted');
        }
    }
```

Figure 77: Code Snippet - User Controller - Delete User

Gym-Managment-System - CoachController.php

```php
/**
     * This function handles the creation of a new coach record.
     * It first checks if an image file has been uploaded in the request.
     * If an image is provided, it is saved to the server with a unique name.
     * If no image is uploaded, a default image ('Client.Png') is assigned.
     */
    public function store(CoachRequest $request)
    {
        $img = $request->file('profile_image');
        if ($img != null) :
            $imageName = time() . rand(1, 200) . '.' . $img->extension();
            $img->move(public_path('imgs//' . 'coaches'), $imageName);
        else :
            $imageName = 'Client.Png';
        endif;

        $coach = new Coach();
        $coach->name = $request->input('name');
        $coach->email = $request->input('email');
        $coach->gender = $request->input('gender');
        $coach->password = Hash::make($request->input('password'));
        $coach->description = $request->input('description');
        $coach->profile_image = $imageName;
        $coach->assignRole('coach');
        $isSaved = $coach->save();

        return redirect()->route('coaches.index');
    }
```

Figure 78: Code Snippet - Coach Controller - Create Coach

Gym-Managment-System - CoachController.php

```php
/*
    This function updates the information of a coach based on the provided input from the request.
    It handles the upload of a new profile image, if provided, and deletes the previous image
    */
    public function update($id, UpdateCoachRequest $request)
    {
        $formDAta = request()->all();
        $coach = Coach::find($id);
        $previousImage = $coach->profile_image;
        $img = $request->file('profile_image');
        if ($img != null) {
            $imageName = time() . rand(1, 200) . '.' . $img->extension();
            $img->move(public_path('imgs/coaches'), $imageName);

            // Delete the previous image from storage if it exists
            if ($previousImage !== 'Client.Png' && file_exists(public_path('imgs/coaches/' . $previousImage))) {
                unlink(public_path('imgs/coaches/' . $previousImage));
            }
        } else {
            // If no new image is uploaded, keep the previous image
            $imageName = $previousImage;
        }
        $coach->name = $request->name;
        $coach->password = Hash::make($request->input('password'));
        $coach->description = $request->description;
        $coach->profile_image = $imageName;
        $isSaved = $coach->save();
        return redirect()->route('coaches.index');
    }
```

Figure 79: Code Snippet - Coach Controller - Update Coach

Gym-Managment-System - CoachController.php

```php
/**
     * This function handles the updating of a coach's profile information.
     */
    public function updateProfile(Request $request)
    {
        $coachID = $request->id;

        $validated = $request->validate([
            'name' => 'required|max:50|min:3',
            'email' => ['required', 'string', 'email', 'max:255', 'unique:coaches,email,' . $coachID],
            'description' => 'required|max:50|min:10',
        ]);

        $oldimg = $request->oldimg;

        if ($request->hasFile('profile_image')) {
            $request->validate([
                'profile_image' => 'image | mimes:jpg,jpeg,png',
            ]);

            $imageName = time() . '.' . $request->file('profile_image')->extension();
            $request->file('profile_image')->move(public_path('imgs//' . 'coaches'), $imageName);

            DB::table('coaches')->where('id', '=', $coachID)->update(['profile_image' => $imageName]);

            if ($oldimg != "Client.png") {
                // to delete old image
                if (file::exists(public_path('imgs//' . 'coaches/' . $oldimg))) {
                    file::delete(public_path('imgs//' . 'coaches/' . $oldimg));
                }
            }
        }

        Coach::find($coachID)->update([
            'name' => $validated['name'],
            'email' => $validated['email'],
            'description' => $validated['description'],
        ]);
        return redirect()->route('dashboard');
    }
```

Figure 80: Code Snippet - Coach Controller – Update Coach Profile

Gym-Managment-System - CoachController.php

```php
/**
     * This function handles the process of updating a coach's password.
     */
    public function updatePassword(Request $request)
    {
        $coachId = auth('coach')->id();

        $data = $request->validate([
            'newpassword' => 'required|min:6',
            'oldpassword' => 'required|min:6'
        ]);

        $userPassword = auth('coach')->user()->password;

        $newPassword = Hash::make($data['newpassword']);

        if (Hash::check($data['oldpassword'], $userPassword)) {     // check if enter old password correct or not

            DB::table('coaches')->where('id', '=', $coachId)->update(['password' => $newPassword]);
            return redirect()->route('dashboard');
        } else {
            $msg = 'الرجاء إدخال كلمة المرور القديمة بشكل صحيح';
            return view('profile.editPassword', ['msg' => $msg]);
        }
    }
```

Figure 81: Code Snippet - Coach Controller - Update Coach Password

Gym-Managment-System - CoachController.php

```php
/**
     * This function retrieves and displays the details of a specific coach based on the provided coach ID.
     */
    public function show($id)
    {
        $coach = Coach::findOrFail($id);
        return view('coaches.show', ['coach' => $coach]);
    }
```

Figure 82: Code Snippet - Coach Controller - Show Single Coach

Gym-Managment-System - CoachController.php

```php
/**
     * This function is responsible for rendering the index view for coaches.
     * It begins by checking if the currently authenticated user has the 'admin' role.
     * It also retrieves the gender of the authenticated user.
     */
    public function index()
    {
        $isAdmin = auth()->user()->hasRole('admin');
        $gender = auth()->user()->gender;

        if ($isAdmin) {
            // Check the gender of the admin and filter users accordingly
            if ($gender === 'male') {
                $coaches = Coach::where('gender', 'male')->get();
            } elseif ($gender === 'female') {
                $coaches = Coach::where('gender', 'female')->get();
            }
        } else {
            // Non-admin users will see all coach
            $coaches = Coach::all();
        }

        return view(
            'coaches.index',
            [
                'coaches' => $coaches
            ]
        );
    }
```

Figure 83: Code Snippet - Coach Controller - View All Coachs

Gym-Managment-System - CoachController.php

```php
// This function handles the deletion of a coach's record.
    public function destroy($id)
    {

        $coach = Coach::find($id);
        $checkSession = CoachSession::where('coach_id', $id)->first();

        if ($checkSession == null) {
            $coach->trainingSessions()->detach();
            $coach->delete();
            return to_route('coaches.index')
                ->with('success', 'Coach deleted successfully');
        } else {
            return redirect()->route('coaches.index')->with('errorMessage', 'cannt be deleted');

        }
    }
```

Figure 84: Code Snippet - Coach Controller - Delete Coach

Gym-Managment-System - TrainingPackageController.php

```php
/**
     * This function is responsible for storing a new training package in the database.
     */
    public function store(StorePackageRequest $request)
    {

        $image = $request->file('image');

        if ($image != null) :
            $imageName = time() . rand(1, 200) . '.' . $image->extension();
            $image->move(public_path('imgs//' . 'gym'), $imageName);
        else :
            $imageName = 'Client.Png';
        endif;
        $package = new Package();
        $package->name = $request->input('name');
        $package->price = $request->input('price');
        $package->image = $imageName;
        $package->description = $request->input('description');
        $package->save();
        return to_route('trainingPackages.index');
    }
```

Figure 85: Code Snippet - Package Controller - Create Package

Gym-Managment-System - TrainingPackageController.php

```php
/**
     * This function is responsible for updating an existing training package in the database.
     */
    public function update($package_id, StorePackageRequest $request)
    {
        $package = Package::findOrFail($package_id);

        $image = $request->file('image');

        if ($image != null) :
            $imageName = time() . rand(1, 200) . '.' . $image->extension();
            $image->move(public_path('imgs//' . 'gym'), $imageName);
        else :
            $imageName = 'Client.Png';
        endif;

        $package->name = $request->input('name');
        $package->price = $request->input('price');
        $package->image = $imageName;
        $package->description = $request->input('description');
        $package->save();
        return to_route('trainingPackages.show', ['package' => $package])
            ->with('success', 'Package Updated Successfully');
    }
```

Figure 86: Code Snippet - Package Controller - Update Package

Gym-Managment-System - TrainingPackageController.php

```php
/**
     * This function is used to display the details of a specific training package.
     */
    public function show(Package $Package)
    {
        $id = $Package->id;
        $package_id = BuyPackage::find($id);
        return view('trainingPackages.show', ['package' => $Package, 'package_id' => $package_id]);
    }
```

Figure 87: Code Snippet - Package Controller - Show Single Package

Gym-Managment-System - TrainingPackageController.php

```php
/**
     * This function retrieves all training packages from the database and prepares them for display.
     */
    public function index()
    {
        $packageCollection = Package::all();
        return view('trainingPackages.index', ['packageCollection' => $packageCollection]);
    }

```

Figure 88: Code Snippet - Package Controller - View All Packages

Gym-Managment-System - TrainingPackageController.php

```php
/**
     * This function is responsible for deleting a training package from the database.
     */
    public function destroy(Package $package)
    {
        $id = $package->id;
        $package_id = BuyPackage::where('package_id', $id)->first();

        if ($package_id == null) {
            $package->delete();
            return to_route('trainingPackages.index')
                ->with('success', 'package deleted successfully');
        } else {
            return redirect()->route('trainingPackages.index')
                ->with('errorMessage', 'cannt be deleted');
        }
    }
```

Figure 89: Code Snippet - Package Controller - Delete Package

Gym-Managment-System - TrainingSessionController.php

```php
/**
     * This function is responsible for creating a new training session based on the data provided in the $request.
     */
    public function store(TrainingSessionRequest $request)
    {

        $start = $request['started_at'];
        $end = $request['finished_at'];
        $end = $request['finished_at'];
        $selectedDays = implode(',', $request->input('day'));


        if ($request->has('coach_id')) {

            $newSession = new TrainingSession();
            $newSession->name = $request->input('name');
            $newSession->days = json_encode($selectedDays);
            $newSession->started_at = $start;
            $newSession->finished_at = $end;
            $newSession->save();
            foreach ($request->coach_id as $coach) {
                CoachSession::create(
                    array(
                        'training_session_id' => $newSession['id'],
                        'coach_id' => $coach,
                    )
                );
            }
            return redirect()->route('sessions.index');
        } else {
            return Redirect::back()->withErrors(['msg' => 'حدث خطأ في إضافة حصة التدريبية']);
        }
    }
```

Figure 90: Code Snippet - Session Controller - Create Session

Gym-Managment-System - TrainingSessionController.php

```php
/**
     * This function is responsible for updating the details of a training session based on the provided $id.
     */
    public function update($id)
    {
        $formDAta = request()->all();

        $start = $formDAta['started_at'];
        $end = $formDAta['finished_at'];

        if ($id) {
            $session = TrainingSession::find($id)->update($formDAta);

            return redirect()->route('sessions.index');
        } else {
            return Redirect::back()->withErrors(['msg' => 'time overlap ,choose another time']);
        }
    }

```

Figure 91: Code Snippet - Session Controller - Update Session

Gym-Managment-System - TrainingSessionController.php

```php
/**
     *  This function is responsible for displaying detailed information about a specific training session,
     *  including the list of users who attend the session based on their gender.
     */
    public function show($sessionID)
    {
        $isWeb = auth()->guard('web')->check();
        $isCoach = auth()->guard('coach')->check();
        if ($isWeb) {
            $gender = auth()->user()->gender;
        } else {
            $gender = auth('coach')->user()->gender;
        }
        $session = TrainingSession::findOrFail($sessionID);
        $users = $session->users->filter(function ($user) use ($gender) {
            return $user->gender === $gender;
        });

        $daysFromDatabase = json_decode($session->days);
        $daysArray = explode(",", $daysFromDatabase); // تقسيم النص إلى مصفوفة
        $daysToShow = implode(", ", $daysArray);
        $session->days = $daysToShow;

        return view('sessions.show', ['session' => $session, 'users' => $users]);
    }
```

Figure 92: Code Snippet - Session Controller - Show Single Session

Gym-Managment-System - TrainingSessionController.php

```php
/**
     * This function is responsible for recording attendance for users in a specific training session based on the provided input.
     */
    public function attend(Request $request, $trainingSession)
    {
        $session = TrainingSession::findOrFail($trainingSession);

        foreach ($request->input('users') as $userId => $days) {
            foreach ($days as $day => $isChecked) {
                if ($isChecked === 'on') {
                    Attendee::where('user_id', $userId)->decrement('remaining_sessions');
                    $attendance = new Attendance();
                    $attendance->attendance_date = now();
                    $attendance->attendance_time = now();
                    $attendance->user_id = $userId;
                    $attendance->training_session_id = $session->id;
                    $attendance->save();
                    $user = User::find($userId);
                    $user->attendances()->save($attendance);
                }
            }
        }
        return redirect()->route('sessions.index');
    }
```

Figure 93: Code Snippet - Session Controller - Attend Session

Gym-Managment-System - TrainingSessionController.php

```php
/**
     * This function is responsible for displaying a list of training sessions based on the user's role and authorization.
     */
    public function index()
    {

        $isWeb = auth()->guard('web')->check();
        if ($isWeb) {
            $roleAdmin = auth()->user()->hasRole('admin');
            $roleClient = auth()->user()->hasRole('client');
            $gender = auth()->user()->gender;
            if ($roleAdmin) {
                $sessions = TrainingSession::whereHas('coaches', function ($query) use ($gender) {
                    $query->where('gender', $gender);
                })->get();
                foreach ($sessions as $session) {
                    $daysFromDatabase = json_decode($session->days);
                    $daysArray = explode(",", $daysFromDatabase); // تقسيم النص إلى مصفوفة
                    $daysToShow = implode(", ", $daysArray);
                    $session->days = $daysToShow;
                }
                $coaches = Coach::where('gender', $gender)->get();
            } elseif ($roleClient) {
                $user = Auth::user();
                $attendance = User::with('attendances.trainingSessions')->find($user->id);
                $sessions = $attendance->attendances->pluck('trainingSessions');
                $coaches = collect();

                foreach ($sessions as $session) {
                    $coaches = $coaches->merge($session->coaches);
                }
                return view('sessions.index', [
                    'sessions' => $sessions,
                    'coaches' => $coaches
                ]);
            }
            return view('sessions.index', [
                'sessions' => $sessions,
                'coaches' => $coaches
            ]);
        } else {
            $isCoach = auth()->guard('coach')->check();
            $roleCoach = auth('coach')->user()->hasRole('coach');
            if ($roleCoach) {
                $coachId = auth('coach')->user()->id;
                $coach = Coach::find($coachId);
                $sessions = $coach->trainingSessions;
                return view('sessions.index', [
                    'sessions' => $sessions,
                ]);
            }
        }
    }
```

Figure 94: Code Snippet - Session Controller - View All Session

Gym-Managment-System - TrainingSessionController.php

```php
/**
     * This function is responsible for deleting a training session with the provided $id.
     */
    public function destroy($id)
    {
        $session = TrainingSession::find($id);

        $checkSession = CoachSession::where('training_session_id', $id)->first();
        $checkAttendence = Attendance::where('training_session_id', $id)->first();

        if ($checkAttendence == null) {

            $session->coaches()->detach();
            $session->delete();

            return to_route('sessions.index')
                ->with('success', 'sessions deleted successfully');
        } else {
            return redirect()->route('sessions.index')
                ->with('errorMessage', 'cannt be deleted');
        }
    }
```

Figure 95: Code Snippet - Session Controller - Delete Session

Gym-Managment-System - PaymentController.php

```php
/**
     * This store function is responsible for creating a new purchase of a training package and updating the user's training sessions.
     */
    public function store(Request $requestObj)
    {
        $requestData = $requestObj->all();
        $package = DB::table('training_packages')->where('id', $requestObj->get('package_id'))->first();
        $user_id = $requestObj->user_id;
        $package_id = $requestObj->package_id;
        $session_id = $requestObj->session_id;

        if ($user_id == null || $package_id == null) {
            return Redirect::back()->withErrors(['message' => 'complete your data']);
        } else {

            BuyPackage::create([
                'price' => $package->price,
                'number_of_sessions' => 35,
                'remaining_sessions' => 30,
                'package_id' => $package_id,
                'name' => $package->name,
                'user_id' => $user_id,

            ]);
            user_sessions::create([
                "training_session_id" => $session_id,
                "user_id" => $user_id,
            ]);
            return to_route('buyPackage.index');
        }
    }
```

Figure 96: Code Snippet - Buy Package Controller - Buy Package

Gym-Managment-System - BuyPackageController.php

```php
/**
     * This function is responsible for displaying the details of a purchased package.
     */
    public function show(BuyPackage $Package)
    {
        return view('buyPackage.show', ['package' => $Package]);
    }
```

Figure 97: Code Snippet - Buy Package Controller - Show Single Purchased Package

Gym-Managment-System - BuyPackageController.php

```php
/**
     * This function is responsible for displaying a list of purchased packages.
     * It includes pagination and filters the displayed packages based on the user's role and gender.
     */
    public function index()
    {
        Paginator::useBootstrapFive();
        $boughtPackageCollection = BuyPackage::paginate(10);
        $isAdmin = auth()->user()->hasRole('admin');
        $gender = auth()->user()->gender;

        if ($isAdmin) {
            if ($gender === 'male') {
                $boughtPackageCollection = BuyPackage::whereHas('user', function ($query) {
                    $query->where('gender', 'male');
                })->paginate(10);
            } elseif ($gender === 'female') {
                $boughtPackageCollection = BuyPackage::whereHas('user', function ($query) {
                    $query->where('gender', 'female');
                })->paginate(10);
            }
        } else {
            $boughtPackageCollection = BuyPackage::where('user_id', auth()->user()->id)->get();
        }
        return view('buyPackage.index', ['boughtPackageCollection' => $boughtPackageCollection]);
    }
```

Figure 98: Code Snippet - Buy Package Controller - View All Purchased Packages

Gym-Managment-System - RevenueController.php

```php
/**
     * This function is responsible for displaying detailed information about a purchased package,
     * typically used in an admin panel to view the details of a specific purchase.
     */
    public function show($boughtPackageID)
    {
        $boughtPackage = BuyPackage::findOrFail($boughtPackageID);
        return view('revenue.show', ['boughtPackage' => $boughtPackage]);
    }
```

Figure 99: Code Snippet - Revenue Controller - Show Single Revenue

Gym-Managment-System - RevenueController.php

```php
/**
     * This function is responsible for displaying a list of purchased packages,
     * typically used in an admin panel to track revenue and user purchases.
     */
    public function index()
    {
        $isAdmin = auth()->user()->hasRole('admin');
        $gender = auth()->user()->gender;

        if ($isAdmin) {
            if ($gender === 'male') {
                //               $boughtPackages = BuyPackage::all();
                $boughtPackages = BuyPackage::whereHas('user', function ($query) {
                    $query->where('gender', 'male');
                })->paginate(10);
            } elseif ($gender === 'female') {
                $boughtPackages = BuyPackage::whereHas('user', function ($query) {
                    $query->where('gender', 'female');
                })->paginate(10);
            }
        }

        return view('revenue.index', data: [
            'boughtPackages' => $boughtPackages,
        ]);
    }
```

Figure 100: Code Snippet - Revenue Controller - View All Revenues

Gym-Managment-System - RevenueController.php

```php
/**
     * This function is responsible for deleting a purchased package record,
     * usually used in an admin panel to remove a specific purchase from the system.
     */
    public function destroy($boughtPackageID)
    {
        BuyPackage::find($boughtPackageID)->delete();
        return redirect()->route('revenue.index');
    }
```

Figure 101: Code Snippet - Revenue Controller - Delete Revenue

Gym-Managment-System - BlogController.php

```php
/**
     * This function appears to be responsible for storing a new blog post.
     */
    public function store(Request $request)
    {
        //
        $validator = Validator::make($request->all(), [
            'title' => 'required|string|min:5',
            'subTitle' => 'required|string|min:10',
            'image' => 'required|file',
            'description' => 'required|string|min:15'
        ]);
        if ($validator->fails()) {
            return Redirect::back()->withErrors($validator->errors());
        }
        $image = $request->file('image');

        if ($image != null) :
            $imageName = time() . rand(1, 200) . '.' . $image->extension();
            $image->move(public_path('imgs//' . 'blogs'), $imageName);
        else :
            $imageName = 'Client.Png';
        endif;

        // handle creator
        $newBlog = new Blog();
        $newBlog->title = $request->input('title');
        $newBlog->subTitle = $request->input('subTitle');
        $newBlog->description = $request->input('description');
        $newBlog->image = $imageName;
        $newBlog->save();
        //redirection to posts.index
        return redirect()->route('blogs.index');
    }
```

Figure 102: Code Snippet - Blogs Controller - Create Blog

Gym-Managment-System - BlogController.php

```php
/**
     * This update function is responsible for updating an existing blog post.
     */
    public function update(Request $request, Blog $blog)
    {
        //
        $validator = Validator::make($request->all(), [
            'title' => 'required|string|min:5',
            'subTitle' => 'required|string|min:10',
            'image' => 'required|file',
            'description' => 'required|string|min:15'
        ]);
        if ($validator->fails()) {
            return Redirect::back()->withErrors($validator->errors());
        }
        $image = $request->file('image');

        if ($image != null) :
            $imageName = time() . rand(1, 200) . '.' . $image->extension();
            $image->move(public_path('imgs//' . 'blogs'), $imageName);
        else :
            $imageName = 'Client.Png';
        endif;

        // handle creator
        $blog->title = $request->input('title');
        $blog->subTitle = $request->input('subTitle');
        $blog->description = $request->input('description');
        $blog->image = $imageName;
        $blog->save();
        //redirection to posts.index
        return redirect()->route('blogs.index');
    }
```

Figure 103: Code Snippet - Blogs Controller - Update Blog

Gym-Managment-System - BlogController.php

```php
/**
     * This show function is responsible for displaying a specific blog post.
     */
    public function show(Blog $blog)
    {
        //
        $blog = Blog::findOrFail($blog->id);
        return view('blogs.show', ['blog' => $blog]);
    }
```

Figure 104: Code Snippet - Blogs Controller - Show Single Blog

Gym-Managment-System - BlogController.php

```php
/**
     * This function appears to be responsible for displaying a list of blogs.
     */
    public function index()
    {
        $blogs = Blog::all();
        return view('blogs.index', ['blogs' => $blogs]);
    }
```

Figure 105: Code Snippet - Blogs Controller - View All Blogs

Gym-Managment-System - BlogController.php

```php
/**
     * This destroy function is responsible for deleting a specific blog post.
     */
    public function destroy(Blog $blog)
    {
        //
        $blog->delete();
        return to_route('blogs.index')
            ->with('success', 'blog deleted successfully');
    }
```

Figure 106: Code Snippet - Blogs Controller - Delete Blog

Gym-Managment-System - AttendanceController.php

```php
/**
     * This function appears to be responsible for displaying attendance records based on the user's role and gender.
     */
    public function index()
    {

        $roleAdmin = auth()->user()->hasRole('admin');
        $gender = auth()->user()->gender;

        if ($roleAdmin) {
            if ($gender === 'male') {
                $attendances = Attendance::whereHas('users', function ($query) {
                    $query->where('gender', 'male');
                })->get();
            } elseif ($gender === 'female') {
                $attendances = Attendance::whereHas('users', function ($query) {
                    $query->where('gender', 'female');
                })->get();
            }
        } else {
            $attendances = Auth::user()->attendances;
        }

        return view('attendance.index', [
                'attendances' => $attendances,
            ]
        );
    }
```

Figure 107: Code Snippet - Attendance Controller - View Users Attendance

Testing

Verification & validation step is considered as one of the most important steps in development life cycle, because it ensures the application determines the requirement of the customer correctly.

Testing operation can be divided into two main parts:

1. **Unit testing**: this testing makes sure that all units of the system work correctly.
2. **Integration testing**: this testing makes sure that all units of the system working together correctly.

Unit Test

Unit testing is done to check whether the individual modules of the source code are working properly. In other words, testing each unit of the application separately by the developer in the developer's environment.

1. Testing `isLoginFormShowed()` function:

   This test ensures that the login form page is accessible. The test sends a GET request to the `/login` route and then asserts that the HTTP response status code is 200 (OK) and that the view being returned is the `auth.login` view. This indicates that the login form is being displayed correctly.

Gym-Managment-System - LoginControllerTest.php

```php
/** @test */
    public function it_can_show_the_login_form()
    {
        $response = $this->get('/login');

        $response->assertStatus(200);
        $response->assertViewIs('auth.login');
    }
```

Figure 108: Code Snippet - Testing – isLoginFormShowed Function

2. Testing `isUserAuthenticated()` function:

   This test verifies that a user can successfully authenticate using valid credentials. It creates a user using the `User::factory(1)->create()->first()` method to generate and save a user to the database. Then, it sends a POST request to the `/login` route with the user's email and password, asserting that the response redirects to the dashboard (configured in the `RouteServiceProvider::DASHBOARD` constant) and that the user is authenticated.

Gym-Managment-System - LoginControllerTest.php

```php
/** @test */
    public function it_can_authenticate_a_user()
    {

        $user = User::factory(1)->create()->first();

        $response = $this->post('/login', [
            'email' => $user->email,
            'password' => '123456789',
        ]);

        $response->assertRedirect(RouteServiceProvider::DASHBOARD);
        $this->assertAuthenticatedAs($user);
    }
```

Figure 109: Code Snippet - Testing – isUserAuthenticated Function

3. Testing `isCredentialsValid()` function:

   This test checks the behavior when a user provides incorrect login credentials. It creates a user in the database as before, then sends a POST request to the `/login` route with incorrect password information. The test asserts that the response redirects to the root URL `root/`, indicating that the login attempt was unsuccessful. It also checks for the presence of session errors related to the email field and ensures that the old input values are retained while checking that the user is not authenticated.

Gym-Managment-System - LoginControllerTest.php

```php
/** @test */
    public function it_redirects_back_with_invalid_credentials()
    {
        $user = User::factory(1)->create()->first();

        $response = $this->post('/login', [
            'email' => $user->email,
            'password' => 'wrong-password',
        ]);

        $response->assertRedirect('/');
        $response->assertSessionHasErrors('email');
        $this->assertTrue(session()->hasOldInput('email'));
        $this->assertFalse(session()->hasOldInput('password'));
        $this->assertFalse(Auth::check());
    }
```

Figure 110: Code Snippet - Testing – isCredentialsValid Function

4. Testing `isUserLogout()` function:

   This test verifies that a user can log out successfully. It creates a user in the database using the `User::factory()->create()` method. The test simulates being logged in as this user by calling `actingAs($user)` and then sends a POST request to the `/logout` route. It asserts that the response redirects to the root URL `root/` and that the user is logged out (`assertGuest()` checks that the current user is a guest).

Gym-Managment-System - LoginControllerTest.php

```php
/** @test */
    public function it_logs_out_a_user()
    {
        $user = User::factory()->create();

        $response = $this->actingAs($user)->post('/logout');

        $response->assertRedirect('/');
        $this->assertGuest();
    }
```

Figure 111: Code Snippet - Testing - isUserLogout Function

5. Testing Result:

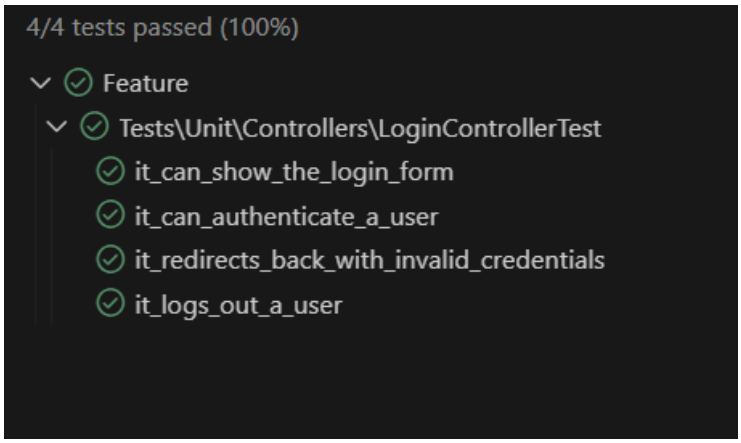


Figure 112: Code Snippet - Testing - Test Result

Integration Test

In this test, we test two or more units of the system units to make sure that they working together correctly.

Table 3: Table of Integration Test Result

Test Case ID	Test Case Objective	Test Case Description	Expected Result
1	Check the interface link between the Login and dashboard of admins, users and coach.	Enter login information for admins, users, coach and click on the Login button.	To be directed to the admins, users and coach dashboard.
2	Check the interface link between the index of view all user, coaches, blogs, services, revenue and session and Delete them Module.	From index of view all user, coaches, blogs, services and session select a user, coaches, blogs, services, revenue or session and click delete button.	Selected users, coaches, blogs, services or session for delete shouldn't appear in index page.
3	Check the interface link between the index of view all user, coaches, blogs, services and session and Edit them Module.	From index of view all user, coaches, blogs, services and session select a user, coaches, blogs, services or session and click edit button.	should appear modify users, coaches, blogs, services and session in index page.
4	Check the interface link between the add users, coaches, blogs, services, session and buy package and index of view all user, coaches, blogs, services, session, and buy package.	Enter users, coaches, blogs, services, session, and buy package information and click on the add button.	To be directed to the index of view all user, coaches, blogs, services, session and buy package.

Conclusions

The development and implementation of our web app for the gym system has been a significant endeavor in addressing the challenges faced by gym owners and members. Throughout this project, we set out to create a user-friendly and efficient tool that would streamline administrative tasks, enhance communication, and provide a better overall experience for gym members.

Through our research and analysis, we identified key pain points and requirements of gym owners and members, allowing us to design a comprehensive solution. The app provides features such as:

1. **Membership management**: The app allows gym owners to manage memberships, including creating new memberships, renewing memberships, and tracking payment history.
2. **Schedule management**: The app allows gym owners to create and manage schedules for classes and trainers.
3. **Booking system**: The app allows members to book classes and trainers online.

Undertaking this project has not only enhanced our technical skills but also deepened our understanding of the challenges faced by gym owners and members and the importance of effective tools and resources for their success. We have learned valuable lessons in project management, user-centered design, and collaboration, which will undoubtedly benefit our future endeavors in the field of IT.

While our project has achieved many of its objectives, there are areas for further improvement and future development. We have identified some insights into additional features that could be integrated, such as:

- **Integration with fitness trackers**: This would allow members to track their fitness progress and share it with their coaches.
- **Personalized recommendations**: This would recommend classes and trainers to members based on their interests and fitness goals.
- **Social media integration**: This would allow members to connect with each other and share their fitness experiences.

These suggestions provide a roadmap for future iterations and enhancements to ensure the app continues to meet the evolving needs of gym owners and members.

In conclusion, our web app for the gym system has demonstrated its potential to revolutionize the way gyms operate. With its user-friendly interface and robust features, the app has the capacity to streamline operations, improve communication, and provide a better overall experience for gym members. We are proud of the achievements and impact of our project and remain committed to its continuous improvement and future growth.

We are grateful for the support of the gym owners and members throughout this project. We are excited to continue working on this project and making it even better in the future.

Future work

The following are the additional features and enhancements that we would like to implement in the future:

- **Mobile version**: We would like to develop a mobile version of the system that will allow users to access the system from their smartphones and tablets. This would make it easier for users to stay connected with the system and track their progress on the go.
- **Exercises collections**: We would like to add a section to the system that will provide users with a variety of exercises to choose from, based on their fitness goals and needs. The section will also provide users with instructions on how to perform each exercise safely and effectively.
- **Improvements and expansions to current features:** We would like to improve the current features of the system by adding new functionality, making the features more user-friendly, and making the features more reliable. We would also like to expand the system by adding new features, such as a vital signs tracker, a sleep tracker, and a stress management tool. These features would help users to improve their overall health and well-being.
- **Improved user experience**: We would like to improve the user experience of the system by making it more intuitive, more responsive, and more visually appealing. We would also like to add more features that allow users to customize the system to their individual needs.

The decision not to include these additional features and enhancements within the scope of this project was primarily driven by resource limitations, including time and technical constraints. Developing and implementing these features would have required additional time, resources, and expertise beyond the scope of our available resources. Furthermore, to ensure the successful delivery of the core functionality, we prioritized the essential features and focused on their implementation and refinement.

Given the opportunity to revisit this project, there are several aspects we would approach differently:

1. **Prioritization of features**: We would conduct a more comprehensive analysis and prioritization exercise at the beginning of the project to determine which features are essential and which could be deferred to future iterations. This would allow us to better allocate our resources and focus on the most impactful aspects of the system.
2. **Resource planning and allocation**: We would carefully assess the available resources, including time, personnel, and technical expertise, to ensure a realistic and achievable scope for the project. This would help us manage expectations and deliver a solution that meets the identified requirements within the given constraints.

By incorporating these improvements into our approach, we can ensure better alignment with stakeholder expectations, efficient use of resources, and the successful implementation of future features and enhancements.

In summary, while we were unable to include certain features and enhancements in this iteration of the system project, they remain essential for the continued growth and effectiveness of the solution. By addressing these aspects in future work, we can further enhance the system's functionality, integration capabilities, and user experience, thereby providing even greater value to users.

Bibliography

[1] Alaasamarah, "Oxygen Gaza - Apps on Google Play," [Online]. Available: https://play.google.com/store/apps/details?id=com.oxygen.gaza.

[2] V. LLC, "Fitness & Bodybuilding - Apps on Google Play," [Online]. Available: https://play.google.com/store/apps/details?id=softin.my.fast.fitness.

[3] L. F. Group, "Home Workout - No Equipment - Apps on Google Play," [Online]. Available: https://play.google.com/store/apps/details?id=homeworkout.homeworkouts.noequipment.

[4] Wikipedia, "Waterfall Model," [Online]. Available: https://en.wikipedia.org/wiki/Waterfall_model.

[5] Wikipedia, "Microsoft Word," [Online]. Available: https://en.wikipedia.org/wiki/Microsoft_Word.

[6] Wikipedia, "Microsoft Project," [Online]. Available: https://en.wikipedia.org/wiki/Microsoft_Project.

[7] Wikipedia, "Diagrams.net," [Online]. Available: https://en.wikipedia.org/wiki/Diagrams.net.

[8] Wikipedia, "Figma (software)," [Online]. Available: https://en.wikipedia.org/wiki/Figma_(software).

[9] Wikipedia, "Visual Studio Code," [Online]. Available: https://en.wikipedia.org/wiki/Visual_Studio_Code.

[10] Wikipedia, "Laravel," [Online]. Available: https://en.wikipedia.org/wiki/Laravel.

[11] Wikipedia, "JavaScript," [Online]. Available: https://en.wikipedia.org/wiki/JavaScript#Web_libraries_and_frameworks.

[12] Wikipedia, "Bootstrap," [Online]. Available: https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework).

[13] Wikipedia, "CSS," [Online]. Available: https://en.wikipedia.org/wiki/CSS.

[14] Wikipedia, "HTML5," [Online]. Available: https://en.wikipedia.org/wiki/HTML5.

[15] Wikipedia, "MVC Pattern," [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

[16] Wikipedia, "Entity Relationship Model," [Online]. Available: https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model.

[17] Wikipedia, "Gantt chart," [Online]. Available: https://en.wikipedia.org/wiki/Gantt_chart.

[18] Wikipedia, "System requirements," [Online]. Available: https://en.wikipedia.org/wiki/System_requirements.

[19] Wikipedia, "Use Case," [Online]. Available: https://en.wikipedia.org/wiki/Use_case.

[20] Wikipedia, "Use case diagram," [Online]. Available: https://en.wikipedia.org/wiki/Use_case_diagram.

[21] L. DiBiase, "The Future of Fitness: How Digital Experiences Will Guide the 2020s," 3 February 2021. [Online]. Available: https://www.uscreen.tv/blog/future-of-fitness-digital-experience/.

[22] W. C. Co., "Fitness Industry Market Research Report," 2023. [Online]. Available: https://www.wellnesscreatives.com/research-report/.

[23] Jerónimo García-Fernández, Manel Valcarce-Torrente, Sardar Mohammadi, Pablo Gálvez-Ruiz, "The Digital Transformation of the Fitness Sector: A Global Perspective," July 2022. [Online]. Available: https://www.researchgate.net/publication/361971495_The_Digital_Transformation_of_the_Fitness_Sector_A_Global_Perspective.

[24] T. N. Y. Times, "The Transformation of the Fitness Industry," [Online]. Available: https://www.nytimes.com/2021/10/19/business/fitness-industry-remote.html.

[25] Wikipedia, "Extreme programming," [Online]. Available: https://en.wikipedia.org/wiki/Extreme_programming.

Appendices

Appendix A – Links to System & Temp Accounts

Here is the app Links and Temporary Accounts for Testing Purposes:

- Admin View:

  Link: http://gym-managementsystem-b49d95e24d17.herokuapp.com/login

  Username: admin@admin.com

  Password: 123456789

- User View:

  Link: http://gym-managementsystem-b49d95e24d17.herokuapp.com/home

  Username: mk@clinte.com

  Password: 123456789

Appendix B - Questionnaire

Here you can see the information collected using the questionnaires:

- Questionnaire Link:

  https://drive.google.com/drive/folders/1GOVraM4uo3mervX_9UOC3b7S8U4A4TIC?usp=sharing

Appendix C - Glossary of Terms

Here is the glossary of terms:

Table 4: Glossary of Terms Table

Terms	Description
IUG	Islamic University of Gaza
XP Approach	Extreme Programming
MVC Pattern	Model, View, Controller Pattern
ERD	Entity Relationships Diagram
HTTP	Hypertext Transfer Protocol